3-2014

Mitigating Differential Power Analysis Attacks on AES using NeuroMemristive Hardware

Colin R. Donahue

Mitigating Differential Power Analysis Attacks on AES using NeuroMemristive Hardware

by

Colin R. Donahue

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of Master of Science
in Computer Engineering

Supervised by

Associate Professor Dhireesha Kudithipudi
Department of Computer Engineering
Kate Gleason College of Engineering
Rochester Institute of Technology
Rochester, New York
March 2014

Approved by:

Dhireesha Kudithipudi, Associate Professor
Thesis Advisor, Department of Computer Engineering

Marcin Lukowiak, Associate Professor
Committee Member, Department of Computer Engineering

Raymond Ptucha, Assistant Professor
Committee Member, Department of Computer Engineering

Thesis Release Permission Form

Rochester Institute of Technology
Kate Gleason College of Engineering

Title:

The Title of the Thesis

I, Colin R. Donahue, hereby grant permission to the Wallace Memorial Library to reproduce my thesis in whole or part.

Colin R. Donahue

Date

Dedication

To my family, for never giving up hope

Acknowledgments

I want to thank my thesis advisor, Dr. Dhireesha Kudithipudi for giving me the opportunity to work on exciting projects and guiding me throughout my thesis work. I also want to thank Dr. Marcin Lukowiak and Dr. Ray Ptucha for serving on my committee. I would like to thank the rest of the faculty and staff who supported me during my time at RIT.

I would not have been able to make it through the many years at RIT so easily without the presence of my friends. Thank you to Jason Lowden for keeping me from getting too lazy. Thank you to Samantha Kenyon, Jeff Myers, Ben Wheeler, Zack Sigmund, and Matt Kelly for keeping my spirits up in the final stretch of my time at RIT. I also must thank Cory Merkel, Levs Dolgovs, and Yu Kee Ooi for their guidance and spending as much time in the lab as me.

Abstract

Mitigating Differential Power Analysis Attacks on AES using NeuroMemristive Hardware

Colin R. Donahue

Supervising Professor: Dhireesha Kudithipudi

Cryptographic algorithms such as the Advanced Encryption Standard (AES) are vulnerable to side channel attacks. AES was once thought to be impervious to attacks, but this proved to be true only for a mathematical model of AES, not a physical realization. Hardware implementations leak side channel information such as power dissipation. One of the practical SCA attacks is the Differential power analysis (DPA) attack, which statistically analyzes power measurements to find data-dependent correlations.

Several countermeasures against DPA have been proposed at the circuit and logic level in conventional technologies. These techniques generally include masking the data inside the algorithm or hiding the power profile. Next generation processors bring in additional challenges to mitigate DPA attacks, by way of heterogeneity of the devices used in the hardware realizations. Neuromemristive systems hold potential in this domain and also bring new challenges to the hardware security of cryptosystems.

In this exploratory work, a neuromemristive architecture was designed to compute an AES transformation and mitigate DPA attacks. The random power profile of the neuromemristive architecture reduces the correlations between data and power consumption. Hardware primitives, such as neuron and synapse circuits were developed along with a framework to generate neural networks in hardware.

An attack framework was developed to run DPA attacks using different leakage models. A baseline AES cryptoprocessor using only CMOS technology was attacked successfully.

The SubBytes transformation was replaced by a neuromemristive architecture, and the proposed designs were more resilient against DPA attacks at the cost of increased power consumption.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

The advent of side-channel attacks (SCAs) has caused increased focus on the security of cryptosystems [21]. Although block ciphers such as AES were considered completely secure in the past, this is no longer the case [35]. SCAs take an entirely different approach to breaking security compared to traditional algebraic or brute-force attacks. The side channel information emitted from an encryption is statistically analyzed along with plaintexts or ciphertexts in order to find the secret key. When implementing a cryptographic algorithm the power consumption and execution time must be considered along with traditional security parameters such as the key size and encryption mode.

Memristors are the fourth fundamental two terminal passive devices that were realized as thin-film devices in 2008 [48]. Memristors can be used to model biological synapses due to their unique properties [15]. Both synapses and memristors have a weight value that can be changed according to a specific training method. The weight of a memristor is represented by its resistance whereas a synapse has a synaptic weight whose strength is controlled by biological parameters. Previously, artificial neural networks have been designed in hardware using both digital and analog techniques. Synapse circuits that use memristors to store weight values have the potential to be more power and area efficient than previous designs which require registers to hold weight values. The focus of this thesis is on using neuromemristive hardware to increase security against a type of side channel attack called differential power analysis (DPA).

There are two main characteristics of neuromorphic architectures that make them a good

candidate for mitigating DPA attacks. The first is that the computation in neural networks is executed in the analog domain by the neurons and synapses in the system. This introduces power fluctuations on the intermediate nodes in the network that appear to be random. In an overdetermined network some neurons will be unnecessary and their output will be unrelated to the overall output of the network. The increased noise level renders attacks that rely on power consumption correlation less effective. The second characteristic is that retraining a network also changes all the synapse weights in the system which in turn changes the power signature of the network. This implies that the power signature of the algorithm can be changed even when the secret key remains the same.

Numerous DPA attacks have been mounted on FPGAs [28, 42, 23, 11] and smart cards [30, 26]. The susceptibility of hardware implementations to DPA has been investigated thoroughly since Kocher's seminal paper on SCAs [21]. Countermeasures against DPA have also been explored and mainly include masking and hiding the power consumption.

Masking adds or multiplies random numbers to an algorithm's input and intermediate values in order to conceal the computation on the actual values [9]. Masking the critical parts of AES can be difficult [36]. Hiding the algorithm's power profile is another approach to countering DPA. Hiding can be performed either by smoothing the power profile so that every operation uses the same amount of power, or randomizing the power so that the computations appear unrelated to the power consumption. Almost all of the previously proposed techniques use traditional CMOS architectures to protect against DPA.

The eventual advent of memristive devices in the commercial market calls for an investigation into how these devices can be used to increase cryptographic security. The security of cryptosystems is of utmost importance in financial and defense applications. Memristive devices open up new avenues in hardware design and this work explores how those avenues can lead to advances in hardware security that are not possible with current conventional designs.

Chapter 2 gives background information about AES, SCAs, memristors, and neuromorphic designs. Chapter 3 discusses related work in SCA prevention and neuromorphic architectures. Chapter 4 describes both the AES architecture and neuromemristive design. Chapter 5 describes the framework for simulating the designs and running attacks. Chapter 6 presents and analyzes the results. Chapter 7 contains concluding remarks and areas for future work.

1.1 Contributions

- Developed fundamental neural network hardware building blocks in SPICE.
- Designed a framework for generating neural networks in software and synthesize in to hardware.
- Implemented two neuromemristive architectures that perform an AES transformation.
- Custom framework design for DPA mounting on neuromemristive hardware (with and without countermeasures).

Chapter 2

Background

This chapter gives an overview of several key components in this work. Background information is given on AES, SCAs, memristors, and neuromorphic architectures. AES is the target algorithm that was implemented using a neuromemristive architecture. AES was attacked using SCAs, in particular DPA. Memristors are one of the primary components of a neuromemristive architecture and used to represent synapses. A neuromorphic architecure was developed in hardware to perform one of the transformations in AES.

2.1 Advanced Encryption System

The Rijndael block cipher was proposed as an Advanced Encryption Standard candidate by Joan Daemen and Vincent Rijmen. It was accepted as a standard by the National Institute of Standards and Technology in 2001 and is now known simply as AES. AES was designed for simplicity, speed, and code compactness and is made up of three invertible transformations [10]. The layers of AES are the linear mixing layer, non-linear layer, and key addition layer. The algorithm consists of several rounds acting on the intermediate result known as the *state*. In the Rijndael algorithm, the *state* can be 128, 192 or 256 bits. The key size can also be 128, 192, or 256 bits. However, when the AES standard was created the size of the *state* was limited to 128 bits. AES can be used for message authentication and hashing along with its more standard use case of a symmetric key algorithm.

The *state* can be thought of as a rectangular array of bytes with four rows. The number of columns depend on the *state* size being operated on. Fig. 2.1 shows an example encryption.

The plaintext and key are simply increasing numbers while the ciphertext appears to be completely unrelated.

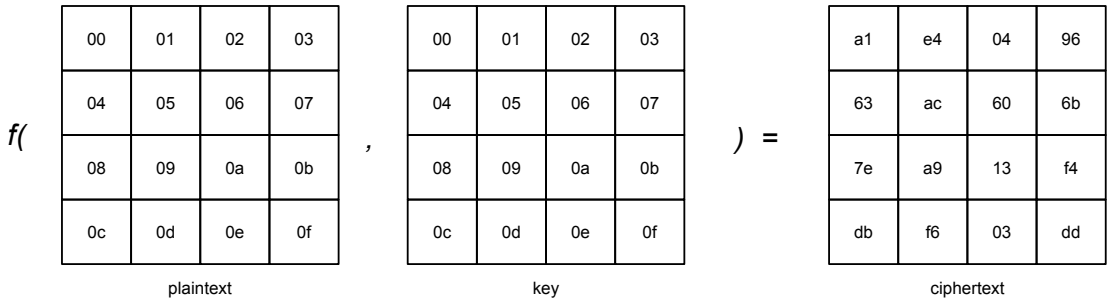


Figure 2.1: Example AES operation. An input plaintext and key yield a ciphertext.

The AES algorithm consists of 10 rounds when a 128-bit key is used. In each round there are four transformations applied to the *state*. The first transformation is SubBytes which is the only non-linear function and simply exchanges each byte in the *state* with another byte. The substitution is defined by finding the multiplicative inverse of the byte in the Galois Field $GF(2^8)$. After the multiplicative inverse is found then an affine transformation on the byte is performed. These two transformations can be computed algorithmically every time a substitution is needed or the entire substitution box (S-box) can be stored as a lookup table in memory. In many software implementations the S-box is implemented as a lookup table since system cache is generally large enough to contain the whole table. There are only 256 possible byte values that can be substituted so only 256 bytes of memory are required. The entire S-box can be stored many times over in cache even in an old processor such as a Pentium I which has 8192 bytes of L1 cache [4].

The next transformation is ShiftRows which shifts each row of the *state* over by a certain number of bytes and is part of the linear mixing layer. An illustration of this is shown in Fig. 2.2. If the total block length is 128 or 192 bits C(1) = 1, C(2) = 2 and C(3) = 3. If the block length is 256 bits then C(3) = 4. The first row of the *state* remains unchanged for all cases. The remaining rows are rotated to the left by 1, 2 and 3 bytes. The third transformation in the linear mixing layer is MixColumns. MixColumns operates on each column of the *state* and can be expressed as a matrix multiplication or an XOR operation

with a fixed polynomial. The fixed polynomial multiplication is shown in Eq. 2.1. The order of the polynomial must be kept below 4, so the result of the multiplication is taken modulo $x^4 + 1$. An illustration of this is shown in Fig. 2.3. Each byte *a* in the original *state* is XOR'd with a polynomial value in *c*.

$$b(x) = (0h03x^3 + x^2 + x + 0h02) \times a(x) \pmod{x^4 + 1} \tag{2.1}$$

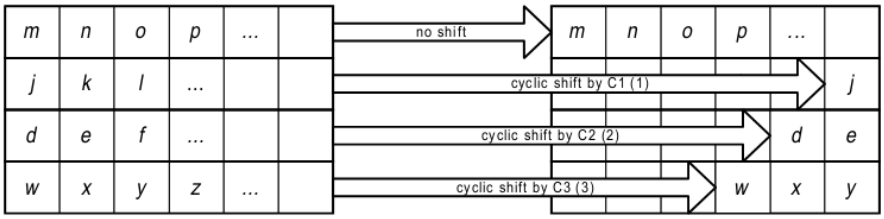


Figure 2.2: ShiftRow on the *state* [10]

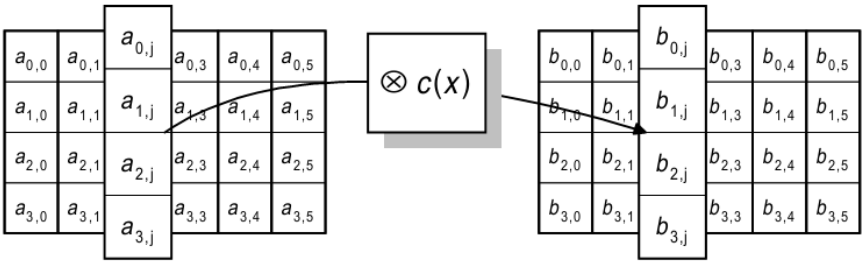


Figure 2.3: MixColumn on the *state* [10]

The final transformation is the round key addition. Each round of the algorithm has a different round key which is based off the secret key and produced by the key schedule. This transformation simply does a bitwise XOR between the round key and state. An illustration of this is shown in Fig. 2.4.

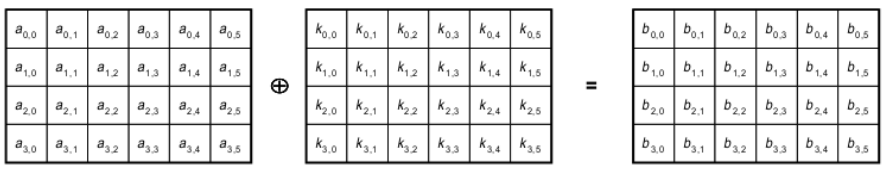


Figure 2.4: AddRoundKey on the *state* [10]

Each round executes these four transformations with the exception of the last round, which does not execute MixColumns. The number of rounds depends on the key size and the block length. The number of rounds varies between 10 and 14. The key size can be 128, 192 or 256 bits. All of these key sizes are currently infeasible to guess using a brute force attack. Assuming an attacker had access to a supercomputer with 100 petaflops of computing power and each encryption took 1000 operations, 1 billion encryptions could be performed per second. Even with that speed, roughly 5×10^{18} years would be needed for a brute force attack.

AES has also been found to be resistant to both linear and differential cryptanalysis as long as at least 4 rounds are used [10]. Thus far, the best attack that recovers the full key is the Biclique cryptanalysis [5]. This analysis still is only four times faster than a regular brute force with a computational complexity of $2^{126.1}$ for 128-bit keys. Even this small reduction only works on the first 7 rounds of an encryption, not the full 10. However, many implementations of AES are susceptible to side channel attacks.

2.2 Side Channel Attacks

Side channel attacks (SCAs) are a unique way of attacking crytographic algorithms in that they do not attack the mathematics of the algorithm. Instead side channel attacks focus on the implementation of algorithms and how they affect the state of objects around them. Traditional cryptographic side channel attacks were first demonstrated by Kocher in 1999 [21]. The first known usage of side channel attacks occured in 1965 [51]. British intelligence officers were able to detect the state of an Egyptian roto-cipher by placing a microphone next to it and listening to the clicks it produced when reset. This allowed the attackers to determine a portion of the secret key without having plaintexts or ciphertexts. Today, there are several forms of SCAs that use timing or power information. The focus of this work is on power attacks.

Both software and hardware implementations of AES leak information that can lead to the secret key. In software implementations the system cache contains information relating to

the instructions to be executed as well as parts of the S-box. If an attacker is able to run a process in a system it is possible to extract information about cache acceses. Tracing the cache accesses with a malignant process gives an attacker information about the data being used in an algorithm as well as the instruction sequence. A malignant process is not always required, sometimes just timing information is sufficient to determine when there was a cache hit or miss. In a hardware implementation of AES the power trace of the circuit can be analyzed. If there are more bits flipped in an operation the current drawn in the system will increase. Similarly, the circuit will consume more power during certain stages of the algorithm and each round of the algorithm can be identified by observing the power trace.

The simplest form of power attacks is Simple Power analysis (SPA). SPA directly analyzes the power consumption of a single execution of a cryptographic operation. The details of encryption algorithms such as DES and AES are fully exposed to the public so an attacker always knows the steps in the algorithm. Oftentimes these implementations are open-sourced as in the case of the popular program OpenSSL [46]. For example, precise current traces of an algorithm can be extracted and show if a branch in code was taken or not taken. This means algorithms where the execution path is data dependent can be easily broken with SPA. However, in most cases the small variation in power consumption is not enough to discover the exact instruction path in hardware implementations. Countermeasures against SPA are also quite simple, the implementation should avoid using branches that depend on keys. This approach can incur significant penalties [29].

DPA uses a different approach. Rather than trying to trace a single execution path, DPA analyzes the effects of different data values being used [20]. For example, in the case of AES the LSB of the S-Box output can be analyzed to see how the power consumption changes for different values. An example distribution of the power consumption taken from 10000 execution traces for AES with the same key but different plaintexts is shown in Fig. 2.5. In this example the instantaneous power was taken at the point in time where the LSB of the *state* was looked up in the S-Box. It is clear that the power consumption of the circuit has a gaussian distribution and there are not obvious discrete sections of

the plot which correspond to unique byte values. In a real system it is not possible to know exactly when a particular operation is occuring. An attacker would instead have to gather power consumption values for an entire execution trace and run many different analyses. Choosing the exact time to look at the power in simulation is much easier since the algorithm is a white box, and gives an idea of what the attacker can do in the best-case situation.

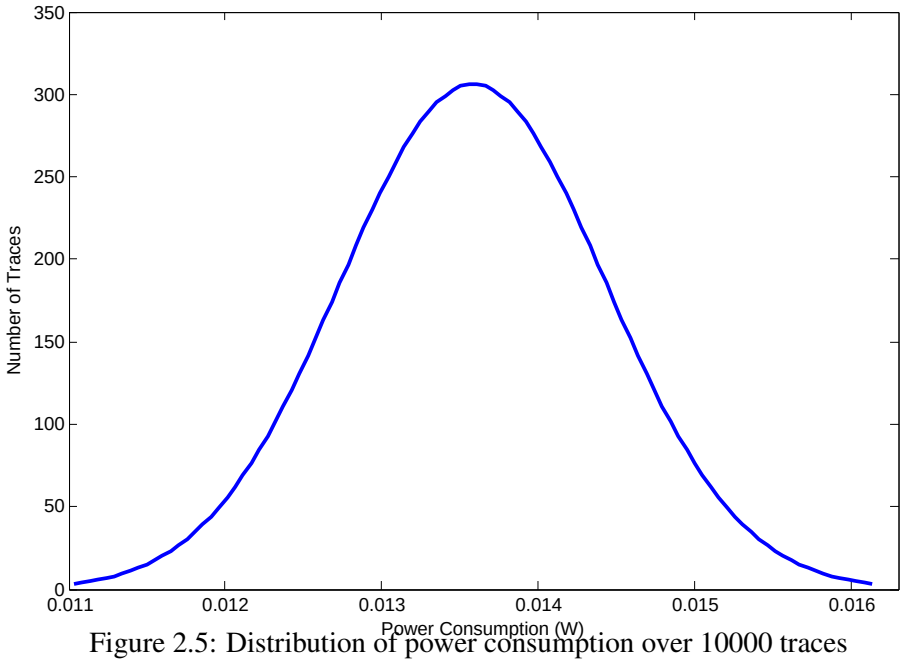


Figure 2.5: Distribution of power consumption over 10000 traces

When gathering power consumption data the attacker also has to keep track of the input plaintexts for each trace. This allows partitioning of the traces based on what data was used in a particular execution. After separating out the traces based on whether the LSB output was a 0 or 1 the distribution in Fig. 2.6 was found. This separation can only be done correctly if the key guess is correct. The average value of these two distributions are noticeably different which implies there is some statistical correlation between the LSB output and the power consumption. Also note that since the two distributions still overlap it would not be possible to predict the output value with only a single measurement or even a few hundred measurements. Hence, obtaining a large number of datapoints is necessary for DPA.

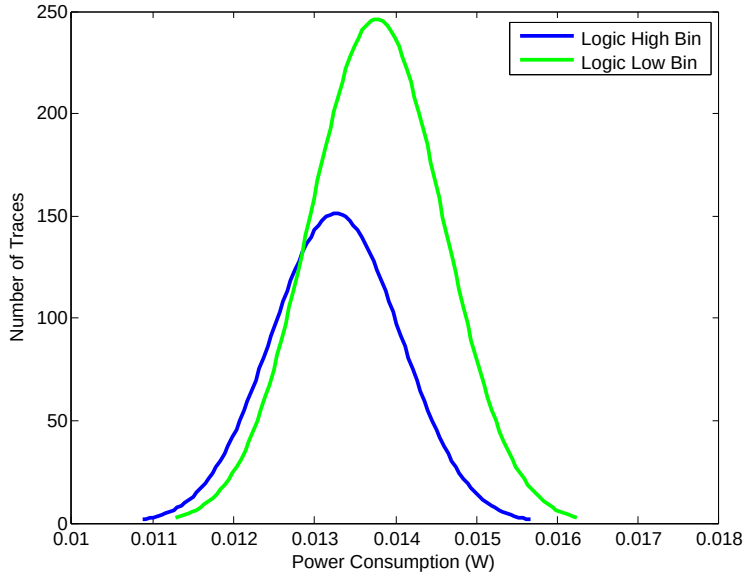


Figure 2.6: Distribution of Power Consumption separated into 0 and 1 outputs with a correct key guess.

When an incorrect key guess is made and the traces are sorted in the same way the distributions look very similar. This can be seen in Fig. 2.7. In this case the average value of each distribution is nearly identical.

A DPA attack on AES focuses on one of two stages in the algorithm. It is assumed the attacker has access to either the plaintext or ciphertext of every execution. If the attacker has access to the plaintext, then the S-box substitution in the first round of execution is analyzed. This is because at that time the plaintext has only been exclusive or'd with the key and is then put through the S-box. After the first round, the plaintext has already been obscured and the attacker cannot determine the input to the S-box. If only the ciphertext is known, then the S-box substitution in the final round of execution can be attacked in a similar fashion.

Regardless of whether a plaintext or ciphertext attack is performed, a selection function must be chosen. This selection function is of the form $D(P, K, b)$. P is the plaintext/ciphertext value, K is the key guess, and b are the target bits. When these inputs are supplied the selection function produces a '0' or '1' output. If the evaluation of P and K causes a change

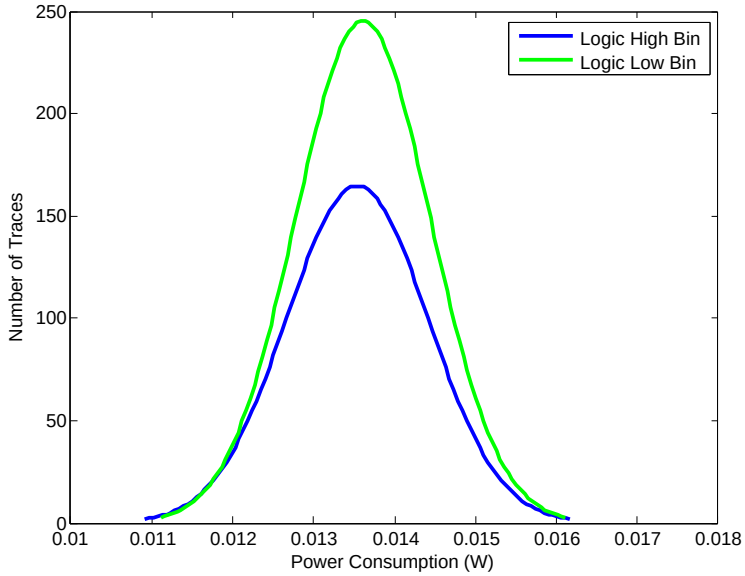


Figure 2.7: Distribution of Power Consumption separated into 0 and 1 outputs with an incorrect key guess.

above some threshold t in b then $D(P, K, b) = 1$. Otherwise, if the change is below that t, $D(P, K, b) = 0$.

The value of the threshold t depends partially on the leakage model that is used to predict the power consumption. The two most popular models are Hamming Distance and Hamming Weight. The Hamming Weight model assumes that a system will consume different amounts of power when an output bit is logic '1' compared to when it is logic '0'. This model puts more importance on static power consumption. Therefore if eight bits were being attacked there would be nine possible Hamming Weights. If the S-box output was predicted to be 01101100 for a particular key guess, the Hamming Weight would be 4 or $HW(01101100) = 4$. If $t = 5$, $D(P, K, b) = 0$. If $t = 3$, $D(P, K, b) = 1$.

Hamming Distance is slightly more complicated since it assumes that if more bits are switching, more power is consumed. This model places more importance on dynamic power consumption. The Hamming Distance model therefore needs to know what an output changed from, not just what it was at a certain point in time. For example, if it was guessed that an output transitioned from 10110111 to 01101100, the Hamming Distance

would be 6. A quick way to evaluate Hamming Distance is to XOR the two values together and find the Hamming Weight, or $HW(O_{n-1} \oplus O_n)$. Once again the selection function will evaluate to different results depending on t.

To perform the attack power traces must be gathered along with the corresponding input values to the algorithm. The input values are randomly generated to ensure a wide variety of power traces. For each power trace that is gathered the selection function is evaluated and sorted into either the A_1 bin or the A_0 bin. If the key guess is incorrect the traces will be sorted into the correct bins with a probability of one-half. This is because although the key guess be far off the actual key, there are only two possible bins to sort into. Even with a random guess the chance of being in the right bin is 50%.

If the key guess is correct the traces will be sorted into the correct bin with a probability closer to one, depending on the threshold chosen. An intelligent attacker will use several different thresholds to see which is the most effective for a particular implementation. The averages of all traces in A_1 and all the traces in A_0 is then computed. The difference between these two averages will be highest when the key is guessed correctly. This is because there is a correlation between the data and power consumption of an algorithm. If the key guess is incorrect there will be a very small difference between the two averages because the traces are essentially sorted pseudorandomly. When attacking AES usually eights bits are guessed at a time so there will be a total of 256 different key guesses. Guessing eight bits at once means there are multiple possible thresholds, and larger differences between averages. When guessing the key a byte at a time, 16 correct guesses need to be made. Even if it is not possible to guesses every byte, each correct guess reduces the search space for the whole key by a factor of 2^8.

AES is vulnerable to DPA any time the power consumption of the system is dependent on a known value and the round key. Any output of a transformation in AES is vulnerable to such an attack except for ShiftRows. ShiftRows doesn't actually perform any logic and can be performed using only wiring. Non-linear transformations can be attacked much more efficiently than linear transformations [20]. Therefore, the ideal transformation to attack is

SubBytes as it is the only one that is non-linear.

Prevention of DPA involves reducing the signal to noise ratio so an attacker gains less information with each sample. One way of reducing this ratio is to randomly generate noise which masks the actual power consumption of the algorithm. An alternative to that technique is to ensure every operation uses the same amount of power by balancing Hamming Weights. If the operations are perfectly balanced every encryption would have the same power consumption so an attacker would not be able to differentiate between encryptions regardless of how many samples are taken.

The goal of these countermeasures are to force the attacker to gather more samples until the amount needed is beyond what is feasible. DPA requires the same key to be used for all measurements so the number of measurements needed has to be less than the number of times a key is used before it is discarded. All of these countermeasures will introduce some degree of performance degradation due to area overhead, speed or power consumed. Leveraging the small size and power consumption of memristors in neuromemristive hardware will minimize the performance costs while still delivering secure computation.

2.3 Memristors

Memristors were first theorized in 1971 by Leon Chua [7]. The well known passive two-terminal circuit elements are the resistor, capacitor and inductor. Memristors are one of the fundamental elements and form a relationship between charge and magnetic flux. Equation 2.2 is the differential equation showing how the change of the magnetic flux linkage and charge depend on the memristance state. Taking an integral with respect to time yields equation 2.3. Equation 2.3 shows how the memristor acts as a resistor when $M(q(t))$ is constant.

$$M(q) = \frac{d\phi_m}{dq} \tag{2.2}$$

$$M(q(t)) = \frac{V(t)}{I(t)} \tag{2.3}$$

A functioning thin-film memristor was fabricated in 2008 by HP Labs [44]. The memristor was made out of a titanium dioxide film and has since been fabricated using several ferroelectric materials [6] [17]. There have also been several variations of memristors, such as spin memristive systems which control the spin of electrons [33] and polymeric memristors [13].

The exact definition of memristors is hotly debated, but a common feature is their pinched hysteresis curve. Chua succinctly stated "If it's pinched it's a memristor" [8]. The resistance level of a memristor can be changed by applying a write voltage to its terminals. Depending on the type of the memristor this resistance behaves in different ways. By the pure definition of a memristor the resistance should always be changing whenever the applied voltage changes. In fabricated devices it takes a sustained voltage above some write threshold to change the resistance of the device [44]. The behavior of fabricated devices is quite useful since the memristors are non-volatile and keep their state when the system using them is powered down. Their resistances can also be read without actually changing the resistance at the same time by applying a read voltage below the write threshold. This means a memristor can be used as a memory device or even as a synapse. The pinched hysteresis curve of a titanium dioxide memristor is shown in Fig. 2.8. This memristor was simulated in SPICE using a model published by Yakopcic et al. [50].

The slope of the straight parts of the curve correspond to two different resistances of the memristor. Once the input voltage goes above the threshold of the memristor (1.088V) it enters a non-linear region where the resistance changes. As can be seen in the Fig. 2.8, the rate at which the resistance changes can be different depending on whether it is increasing or decreasing. In this particular model, the resistance increases faster than it decreases, so the resistance change when a negative voltage is supplied is larger. These differences do not affect the operation of the memristors when voltages are below the threshold, which is how they are used in this work.

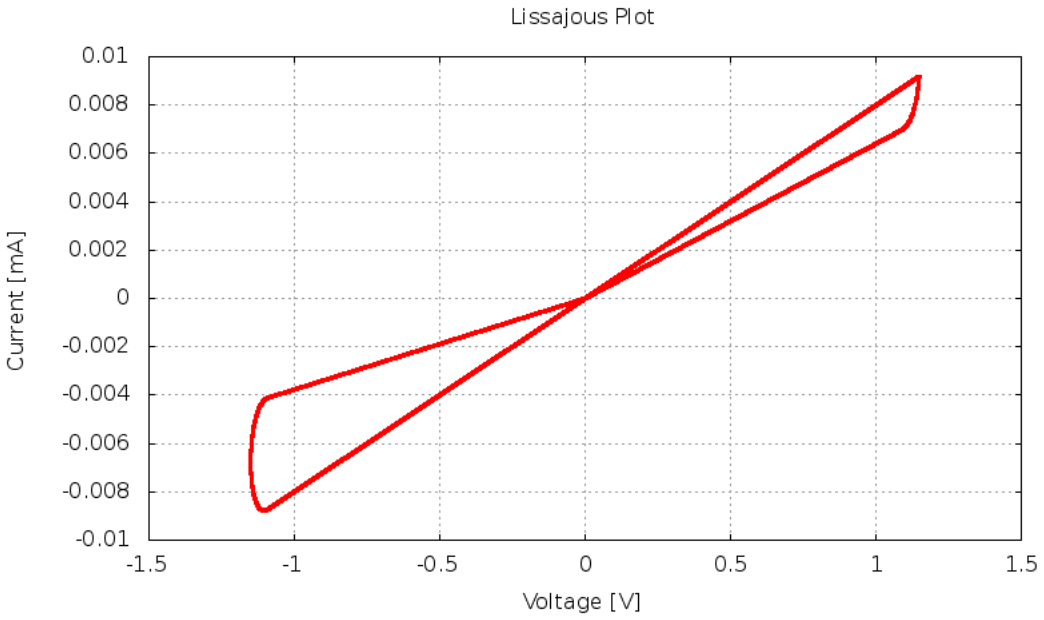


Figure 2.8: Pinched hysteresis curve

One of the current limitations of memristors is that their fabrication is not yet a reliable process. Integrating memristors with CMOS logic also poses new design challenges. In 2012 researchers at the University of Michigan and HRL Laboratories [18] were able to fabricate a memristor/CMOS system that stored a bitmap image. The memristor array was part of a vertical layer that sat above the CMOS control logic.

The advent of memristors allows for the design of architectures previously infeasible in hardware. The similarity between memristors and synapses opens up the possibility of new neuromorphic computing paradigms in hardware. In a brain, neurons are connected to other neurons by synapses which vary in strength. A stronger connection (in the case of electrical synapses) means that more current will pass through and cause a greater voltage change in the next neuron. A memristor with a high conductance value mimics a stronger connection. Memristors can also be placed compactly and consume little power which gives hope that neuromorphic architectures will be able to give the same performance as CMOS in certain applications while consuming less energy and with a lower area cost.

2.4 Neuromorphic Architectures

2.4.1 Neural Network Basics

Artificial neural networks are biologically inspired designs that are capable of doing computation in a unique way. Traditionally computers are designed using the Von-Neumann architecture which has registers, a program counter, memory, and an arithmetic logic unit. One of the primary problems in Von-Neumann architectures is that there is often a memory bottleneck for high performance computing applications. Neural networks differ from traditional architectures since they are modeled to be similar to a biological brain and contain two fundamental building blocks: neurons and synapses. Neurons are connected to each other by synapses and each synapse has a weight associated with it. Each neuron also has an activation function which is activated when the summation of all the input weights is above a certain threshold.

The way neurons and synapses interact is described by Equation 2.4. The output of a neuron is y. The input contributions from other neurons are x_i and the corresponding weight connection between neurons is w_i. An additional bias value b is often included to increase training accuracy. The activation function f is usually a sigmoid, linear, or step function.

$$y = f(\sum_i w_i x_i + b) \tag{2.4}$$

There are three main classifications of neural networks: multilayered, single-layered and recurrent [24]. The neurons in the multilayer networks are only connected to neurons outside their layer. An example multilayer perceptron network architecture is shown in Fig. 2.9. A network with more hidden layers and neurons will be able to perform complex functions with greater accuracy.

A recurrent network contains connections from the outputs to the inputs. Fig. 2.10 shows an example of a recurrent network. A popular type of recurrent network is a Hopfield

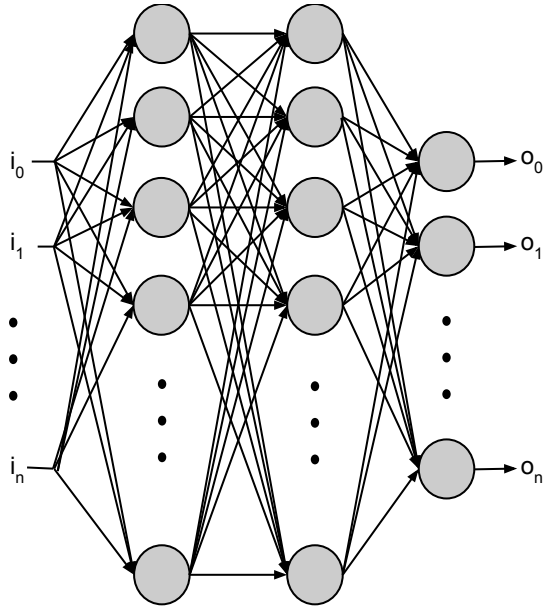


Figure 2.9: Multilayer Perceptron Network. Each neuron is connected to every neuron in the subsequent layer.

network. Hopfield networks can be used as associative memory and image classification and consist of only a single layer where every neuron is connected to every other neuron. The main advantage of recurrent networks is that they can be trained to perform functions on time-varying inputs. They also are designed to more closely approximate the capabilites of a biological brain which has a large number of recurrent connections. Networks such as Echo-State Networks and Liquid State machines are able to perform tasks such as image recognition and audio processing on time-varying data.

In general, neural networks are naturally good at operations with large degrees of parallelism, systems with large numbers of inputs and outputs, and when little is known about how a system should be designed other than inputs and outputs. An important consideration when using neural networks is how to train them. The selection and size of training data greatly affects how fast a network can be trained. Similarly the initial weights of synapses can lead to different training results. Knowing when to stop training can either be decided by establishing a maximum output mean squared error or seeing when the synapse weights

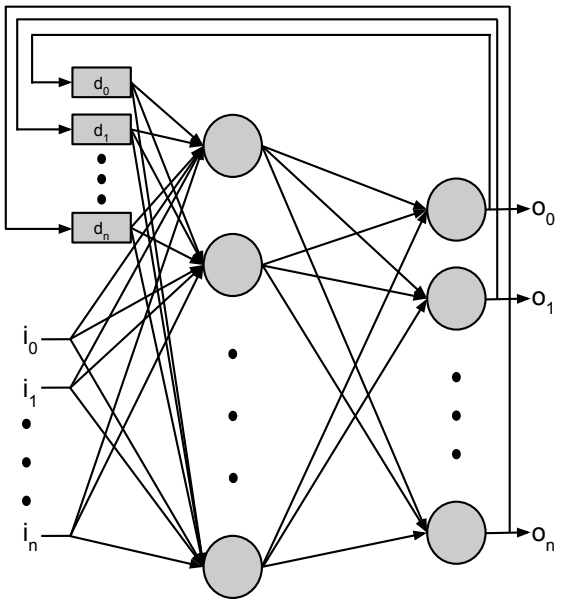


Figure 2.10: Recurrent Network. Each neuron is connected to every neuron in the subsequent layer. The outputs are fed back into the network with a delay parameter. This allows the network to process temporal data.

stop changing in a way that affects the output. The exact time to stop training is somewhat subjective and varies from application to application. Designing the size of a network tends to be a trial and error process and the size needed can not be predicted until several performance tests are done for different sizes of networks.

2.4.2 Network Training

The goal of training a neural network is to reduce some cost function associated with it. In this work, the cost function is the overall accuracy of the network at approximating an AES transformation. The smaller the cost, the more accurate the transformation. In particular, the mean squared error between the network and expected outputs is found. The training method used is supervised, which means sets of inputs and outputs are presented to the network and it tries to infer a function between them.

In most cases, an important aspect of a neural network is its ability to generalize a function so that it can respond well to previously unseen inputs. Input data for training is often

divided into training and testing sets. Training data is used to train the network, and testing data is used to ensure the network will generalize well. For example, a neural network that differentiates between male and female faces shouldn't fail just because it sees a new face. Therefore, it is often important not to overtrain a network and see patterns that do not necessarily exist for all possible inputs. Fig. 2.11 shows an example what can happen when a network is overtrained.

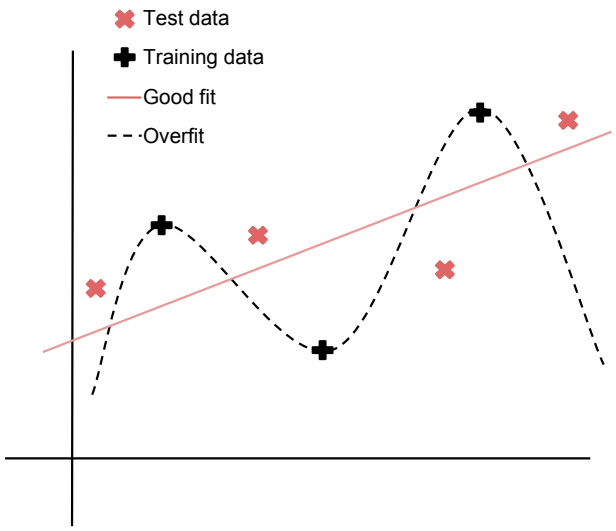


Figure 2.11: Effects of overtraining a neural network.

In this case, a linear approximation of the input would yield good accuracy results for both training and testing data. A higher order function is able to have better performance for just the training data, but when the testing data is added in the performance is much worse.

In this work, it is better to overtrain the network. This is because the inputs to the network are binary and every possible input is known ahead of time. Most importantly, there is not a prohibitively large amount of possible inputs so every one can be presented to the network for training in a reasonable amount of time. A network training to differentiate between human male and female faces could be presented the face of every human, but it would take an incredibly long time to train.

Gradient descent methods are often used to minimize the cost function of a neural network. The gradient descent method works by continually moving in the direction of the negative gradient. Eq. 2.5 shows how x would be updated to find a minimum of $f(x)$.

$$x_{new} = x_{old} - \epsilon f'(x_{old}) \tag{2.5}$$

This method is not guaranteed to find a global minimum for all functions. It is not even guaranteed to find a local minimum for all functions, and has very poor convergence speed as it approaches a minimum. One feature that helps with overall convergence speed is the ϵ term. A larger epsilon term will make the convergence speed faster but it may pass over some local minima due to having a large step size. Oftentimes ϵ is changed dynamically during training depending on how close to the target performance the algorithm is.

The more specific delta rule can be derived from the idea of gradient descent and is shown in Eq. 2.6.

$$\Delta w_{ij} = \epsilon(t_j - y_j) f'(\sum_i w_{ij} x_i) x_i \tag{2.6}$$

This equation governs how the ith input to the jth neuron changes. t_j is the target output of the neuron. $f(x)$ is the activation function of the neuron, and $y_j = f(\sum_i w_{ij} x_i$ which is the output of the jth neuron. w_{ij} is the weigth between the jth neuron and ith input to that neuron. ϵ is the learning rate and controls how fast the weight will change.

The delta rule can be used as-is for a single layer network. For a network with multiple layers it has to be extended using a method called backpropagation. Backpropagation first evaluates the whole network for a particular input sequence, then applies the delta rule for the output layer, the next to last layer, and so on until it reaches the first layer. This is then repeated until the desired accuracy is acheived. Since it is a simple gradient descent method it does not guarantee convergence to a global mininum and can converge slowly.

The method used for training in this work is Levenberg-Marquardt algorithm. This algorithm uses an iterative method similar to backpropagation, but has an additional damping factor that gives fast convergence even when the error gradient is small. Eq. shows the algorithm which must be solved for the new change in parameter vector δ. After each iteration the parameter vector β is updated to $\beta + \delta$.

$$(J^T J + \lambda diag(J^T J))\delta = J^T[y - f(x, \beta)] \tag{2.7}$$

In this equation f is also a vector with entries $f(x_i, \beta)$ where each x_i is an input to the overall neural network function and β contains all the parameters (weights). y is a vector of the expected output y_i for each input combination. The Jacobian matrices J have rows $J_i = \frac{\partial f(x_i,\beta)}{\partial \beta}$. The damping factor λ starts out small when beginning training since it is assumed the parameters are far from the minimum error. If the parameter update yields a smaller error, then λ is reduced more for the next iteration. If the new parameters result in a higher error, the iteration is repeated with an increased λ. The Levenberg-Marquardt algorithm can be stopped when λ is so large that further iterations have no substantial effect on the error.

Neural networks can be trained to emulate the various AES transformations. Non-linear transformations in AES such as SubBytes will require a feedforward neural network with a sigmoid activation function. Simpler transformations such as ShiftRows do not even require CMOS logic so using a neural network would induce unnecessary overhead. Neural networks are often used with analog inputs, whereas AES uses only binary inputs and outputs. This means the outputs of the trained network do not need to exactly equal '0' and '1'. Outputs larger than 0.5 can be classified '1' which reduces the required accuracy of the training algorithm. However, every single output needs to be classified correctly for every input, since a single incorrect bit will propagate through the entire algorithm and make the final output completely unrecognizable.

Retraining a network also changes its synaptic weights which will give it a different power signature even if the same key is used. Changing the power signature of a system will make

it difficult for DPA attacks to be successful since they rely on creating correlations between encryptions.

Chapter 3

Related Work

Since the advent of SCAs on AES there have been many proposals to mitigate their effectiveness. There have been no attempts so far to create countermeasures using a neuromemristive architecture. Neural networks have been used in cryptographic schemes but not in the context of preventing side-channel attacks. A number of neural network designs in hardware using memristors have also been proposed.

3.1 Neuromemristive Designs

Rose et al. [37] explored using memristors to make low-power neuromorphic hardware. A subthreshold CMOS-memristor circuit was developed that used energy on the order of femtojoules. However, the delay of the circuit was quite long, about 500 μs. The network was also not trained to perform a specific task.

Adhikari et al. used a memristor bridge synapse to develop a hardware neural network capable of computing 3-bit parity and recognizing cars in an image [1]. The synapse design used four memristors and could have both positive and negative weight values. Learning methods were also explored and a multilayer network was trained by training each layer separately. The performance of the trained hardware network was similar to that of a software network. The architecture was designed to reduce power consumption by using a pulse based technique so it didn't have to use a large number of input terminals. This design did not use a truely sigmoidal activation function so it cannot be trained to perform the S-Box transformation. Ebong et al. compared CMOS and memristor MOS technologies

in neural network applications based on power usage, area and SNR tolerance [12]. They found that the memristor based network was better in each category.

Yakopcic et al. presented a segmented memristor crossbar array that performed pattern recognition [49]. The crossbar was segmented to improve the energy efficiency of the design and reduce unwanted current paths. The system used one memristor per synapse and was therefore unable to use negative weights.

3.2 Neural Networks in Cryptography

Noughabi and Sadeghiyan [27] explored the potential of using neural networks to generate secure substitution boxes. They found that recurrent neural networks were able to fulfill some of the important properties of substitution boxes such as non-linearity, completeness and strict avalanche. They also attempted to match the AES S-Box but were not successful.

The primary use of neural networks in cryptography has been for a public key exchange protocol [38]. In this use case two neural networks are initialized with randomly chosen uncorrelated weights. The networks are then synchronized with one another by receiving common input values, transferring their output values to one another, and then updating their weights. Two networks actively synchronizing with one another will converge to the same output faster than another network listening to the communication. Increasing the synaptic weight range available increases the security of the algorithm. Three methods have been developed to attack a neural cryptography scheme: geometric, genetic, and probabalistic [19]. However, the computational complexity of each attack rises faster than the complexity of the protocol when the synaptic weight range is increased. Therefore, the attacks have a small chance of success when a proper weight range is chosen.

Stottinger et al. implemented a lightweight neural cryptography scheme and tested the effectiveness of side channel attacks against it [43]. Their implementation did not include any memristors. Instead it used a tree parity machine with components like multiplexers

and adders. It was found that the scheme was resistant against DPA and became more resistant when more internal nodes were added. The system was not completely invulnerable against side channel attacks but it made practical attacks expensive. Their cryptographic scheme also did not have any non-linear functions such as a substitution box which would make it more vulnerable to DPA attacks.

3.3 DPA Countermeasures

3.3.1 Masking

The *tower field method* uses both additive and multiplicative masks and performs the S-Box inversion operation in $GF(2^2)$ [31]. This was implemented in hardware and found to increase the critical path of the circuit by five times and also increase the area by about 1.5 times. The *table re-computation method* uses boolean and arithmetic masks along with a masked lookup table [25]. This approach led to a two-fold increase in the number of execution cycles. The masking techique proposed by Matthieu et al. [36] can provably stop higher-order DPA in software. Their technique was slower than the *table re-computation method* and the *tower field method* for first-order protection. The number of cycles needed for an unmasked implementation was 3×10^3 and this increased to 129×10^3. However, for protecting against higher-order DPA their technique was more efficient in terms of code size and execution cycles.

3.3.2 Hiding

One technique used to completely level a circuit's power consumption is dual rail logic [14, 34]. Guilley et al. [14] used *wave dynamic differential logic* (WDDL), the goal of which is to duplicate a netlist into two parts whose transitions are the opposite of one another. Naturally this approach will double the hardware area since an entire copy of the circuit must be made. The WDDL technique was experimentally tested using FPGAs and found to be effective against DPA but the authors suspect that a more sensitive measurement

platform could still mount a successful attack.

Popp et al. [34] proposed the use of *masked dual-rail precharge logic* (MDPL). This approach avoids the requirement of perfectly balancing the capacitive complementary wires. It accomplishes this feat by masking every value and precharging cells. However, the area of basic cells such NAND and NOR increase by a factor of 4. MDPL was found to have higher energy and area costs as well as slower execution time than approaches such as WDDL. However, it was found to improve DPA resistance without requiring perfectly balanced wires.

A method of power randomization was proposed by Ambrose et al. which injected random code into cryptographic algorithms [3]. Critical blocks in the code which contained cryptographic operations are identified and when these blocks are executed instructions are called which write to random registers. This technique caused an average runtime increase of almost 30% and a 27.1% increase in energy consumption.

An asynchronous, low power substitution box was developed by Wu et al. [47]. The hardware was synthesized onto an FPGA and found to be resistant to DPA. The S-Box was designed with a special self-timing logic with no clock and monotonic transitions. The implementation had lower power consumption than a synchronous equivalent.

An AES implementation that used dual-memories to create a uniform power signature during encryption was proposed by Khedkar [16]. The implementation was tested by using both CMOS and RRAM dual memories. It was resistant to DPA even after 40,000 power measurements and the RRAM architecture consumed less than half the average power as the CMOS architecture.

3.4 Summary

Overall, there have been forays into using neural networks for cryptography but they tend to be less effective than traditional implementations. This is largely due to the fact that

neural networks must be simulated in software on a Von-Neumann machine which is inherently inefficient or designed using large of amounts digital logic. The proposed design uses memristors to model a neural network in analog which is a more natural fit. DPA countermeasures also explicitly add extra masking or hiding hardware whereas a neural network implementation includes randomness that hides the power signature by default.

Chapter 4

Proposed AES Neuromemristive Architecture

This chapter presents an overview of the hardware components which combine together to form the entire architecture. The primary building blocks of an AES implementation are shown in Fig. 4.1. The control unit is a state machine which determines where memory should be read from and written to as well as which transformations should be performed. The key memory stores the secret key value and can also store the expanded round keys. The RAM can store plaintexts, ciphertexts, and intermediated values.

In this work the only transformation that was modified was SubBytes. In the reference implementation all the blocks were designed using traditional CMOS logic. The new SubBytes implementation used a neuromemristive design.

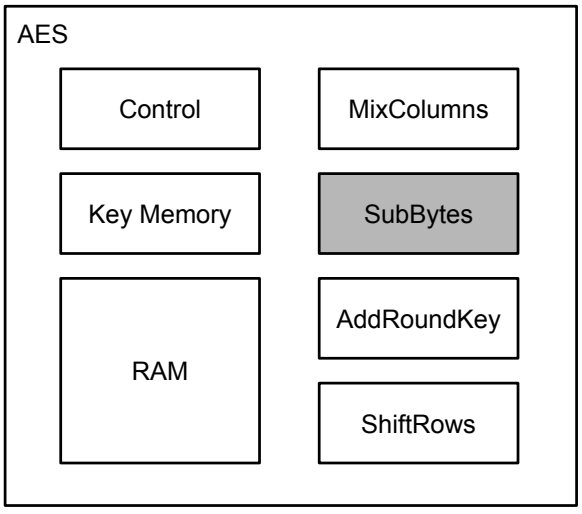


Figure 4.1: The basic hardware blocks of an AES implementation.

A neural network in hardware has two primary building blocks: neurons and synapses. Fig. 4.2 shows how the blocks are connected. Each neuron is connected to many output synapses which are then connected to another neuron. The number of layers and neurons per layer depends on the complexity of the problem the network is being trained to solve.

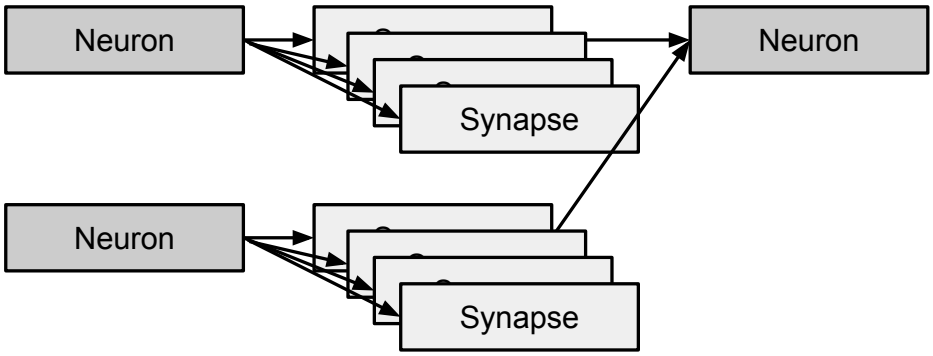


Figure 4.2: The basic hardware blocks of a neural network.

4.1 Neuron Design

The neuron in a neural network has to perform two operations. First, it must sum all of the input contributions from incoming synapses. Then it must apply an activation function to that input and produce an output. In the neuron design the summing is performed by a differential amplifier and the activation function is computed using a differential pair of transistors.

The ideal behavior of the summing circuit would take the positive and negative contributions from synapses (V_+ and V_-) and find the difference and ouput that voltage (V_{out}). This is shown in Eq. 4.1. Fig. 4.3 shows the circuit for a differential amplifer. This circuit's behavior is governed by Eq. 4.2 and can be simplified as shown in Eq. 4.3 where A is the gain. As long as the gain is limited to 1 then Eq. 4.1 and Eq. 4.3 are equivalent.

$$V_{out} = V_+ - V_- \tag{4.1}$$

$$V_{out} = (\frac{R_{in-} + R_{f-}}{R_{in-}})(\frac{R_{f+}}{R_{f+} + R_{in+}})V_+ - (\frac{R_{f-}}{R_{in-}})V_- \tag{4.2}$$

$$\text{if } (\frac{R_{f-}}{R_{in-}}) = (\frac{R_{f+}}{R_{in+}}) \longrightarrow V_{out} = A(V_+ - V_-) \tag{4.3}$$

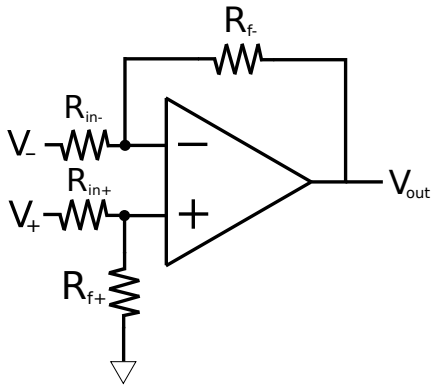


Figure 4.3: Basic differential amplifier using an op-amp

The activation function is computed by the circuit shown in Fig. 4.4. It is essentially a differential pair that is able to have an input between V_{dd} and V_{ss}. The output ports V_+ and V_- are both used by the synapse circuit and allow the output of the neuron to be bipolar. The neuron circuit is designed to have a sigmoidal shape and approximate the *tanh* function.

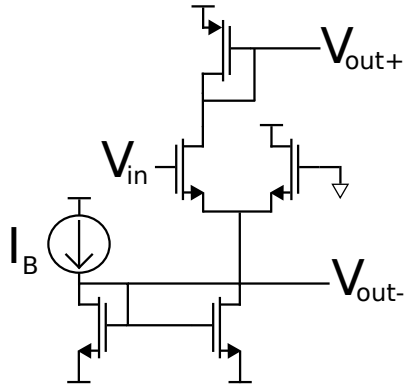


Figure 4.4: A differential pair of transistors that approximates the $tanh$ function

4.2 Synapse Design

The synapses mirror current from the neuron activation function outputs. The neuron and synapse are designed in such a way that the output current to the synapse ranges from I_B to $-I_B$. When $V_{in} = V_{dd}$ in the neuron circuit I_B flows through the left branch of the differential pair. This is then mirrored over to the synapse circuit and multiplied by a factor of 2 because of the double size transistor on the synapse. I_B is also mirrored to V_{in-} on the synapse so the summed current is equal to $2I_B - I_B = I_B$. In the opposite case where $V_{in} = V_{ss}$ the summed current in the synapse is $0I_B - I_B = -I_B$. In the general case where I_x is flowing through the left branch of the neuron the summed current is $2I_x - I_B$.

After the current is summed it then branches depending on the values of the two memristors M_+ and M_-. When M_+ is at its highest conductance value and M_- is at its lowest conductance value the synapse has a value of almost $+1$ since nearly all the current will flow through M_+. In the opposite case where M_- is at its lowest conductance value and M_+ is at its highest the synapse has a value of -1.

A SPICE memristor model [50] was used to test the synapse circuit. The model was fitted to experimental data found by Lu et al. [22]. This memristor has a positive write threshold of $1.088V$, negative write threshold of $-1.088V$, and a resistance range of $125k\Omega$ to $125G\Omega$.

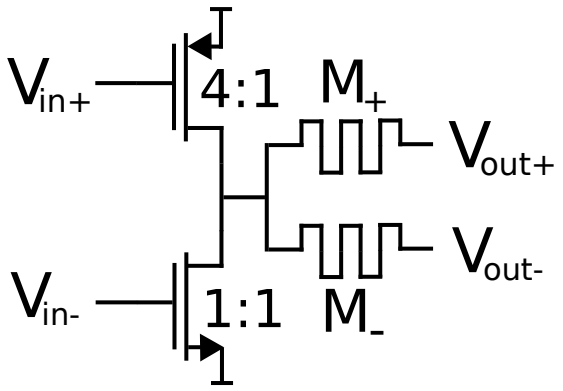


Figure 4.5: The synapse circuit where the weight is controlled by the memristors M_+ and M_-.

The synapse, differential amplifier, and differential pair are then connected together as

shown in Fig. 4.6. Resistors are needed between the synapses and differential amplifier to convert the current output of the synapses to a voltage. Mutiple synapses can be connected to one differential amplifier and share the same load resistors. The two outputs of the differential pair are connected to the inputs of the synapses in the next layer of the network.

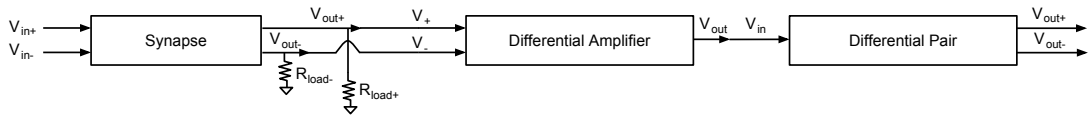


Figure 4.6: The connections made between neural network building block circuits.

4.3 S-Box Design

The S-box function was built using a neural network consisting of the components presented in sections 4.1 and 4.2. The S-box function has eight binary inputs and eight binary outputs. The true difficulty in training a neural network to compute this function is that it has to be accurate for every single input combination, of which there are $2^8 = 256$.

In order to make the training process both faster and parallelizable eight separate networks were used, one for each output. In addition, the security of the function depends on having what seems to be random power consumption. If the user desires the system to be retrained only a single, smaller network needs to be retrained rather than an entire large network. If the user wants retraining to be done between every encryption then the downtime between encryptions will be lower if only an eighth of the S-box function needs to be retrained.

The size of network is larger than what is needed in software for several reasons. In hardware the weight range was limited between -1 and 1, which reduces the number of solutions a network can converge to during training. The hardware neurons and synapses are also not the ideal models that can be found in software which the training algorithm cannot directly

compensate for. Adding extra neurons in the hidden layer also means that some neurons' output may not be directly related to the output of the network which adds additional noise to the system. The initial size of the network was determined by trial-and-error until 100% accuracy could reliably be achieved. Simpler problems such as a 4-bit XOR were tested to see the increase of neurons that would be needed in the transition from software to hardware. The degree to which the size of the S-Box network needed to be increased in hardware could then be found more quickly.

Each individual network had 8 inputs, a layer of 48 hidden neurons and a single output neuron (8:48:1). The connections of this network are shown in Fig. 4.7. In Fig. 4.7 the inputs are simply input voltages ranging from $-1V$ to $+1V$. Each arrow requires a synapse circuit. The hidden and output layers neurons both require a full neuron circuit. The final output also has a comparator circuit which converts voltages greater than $0V$ to $1V$ and voltages less than $0V$ to $0V$. This comparator is needed so that this analog neural network can interface with a digital circuit.

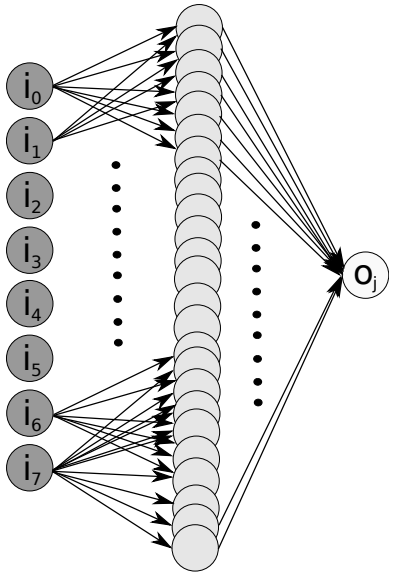


Figure 4.7: The network architecture for a single output o_j. There are 8 total networks, one for each S-box output. The eight inputs are fully connected to the hidden layer of 48 neurons which are in turn all connected to the output neuron.

An alternative architecture was also designed to add even more randomness to the power

consumption. Eight networks with 7 inputs and 2 outputs were also trained and used in a different configuration (7:48:2). Four of the networks were trained assuming the MSB of the input value was '0' and the other four assuming an MSB of '1.' Together these networks can generate the SBox output but only half need to be active at a time. The other half can be fed random noise and their output is ignored. This architecture is shown in Fig. 4.8.

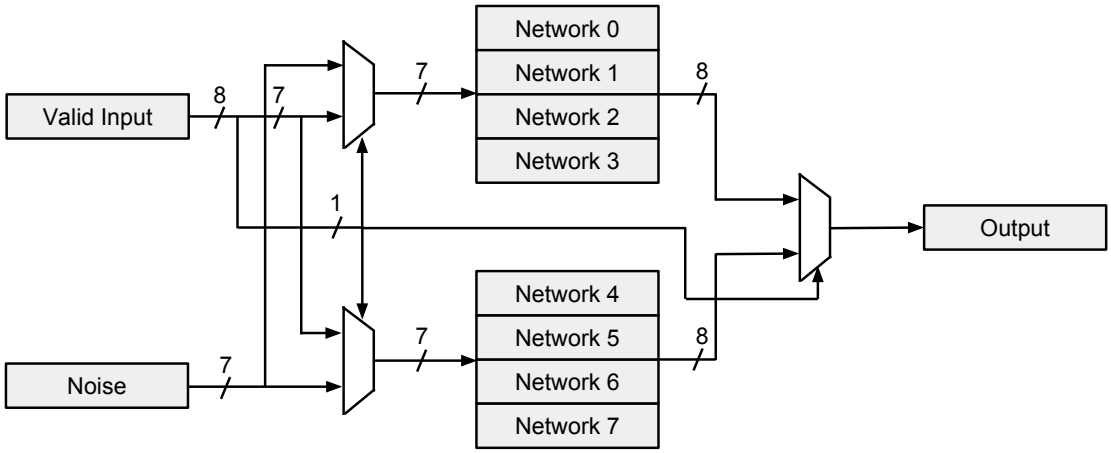


Figure 4.8: The architecture of the neural network with random values added. The MSB of the input controls both the multiplexers.

4.4 AES Architecture

The AES implementation operates on an 128-bit key and 128-bit state and was adapted from the work done in [39]. The unit was designed structurally and consists of a high level encryption unit, a control block, memory, and blocks which perform the four round operations. The control block state machine determines when each operation and round is over and when to read and write to memory. This state machine is shown in Fig. 4.9. AES requires 10 rounds of execution when using a 128-bit key but this value can be changed by the control block since only one round is required to mount a DPA attack.

The state diagram shows how in each transformation all 16 bytes of the *state* are calculated

sequentially before moving on to the next state. After loading the bytes of the *state* the AddRoundKey, SubBytes, ShiftRows and MixColumns transformations are performed before storing the output *state*.

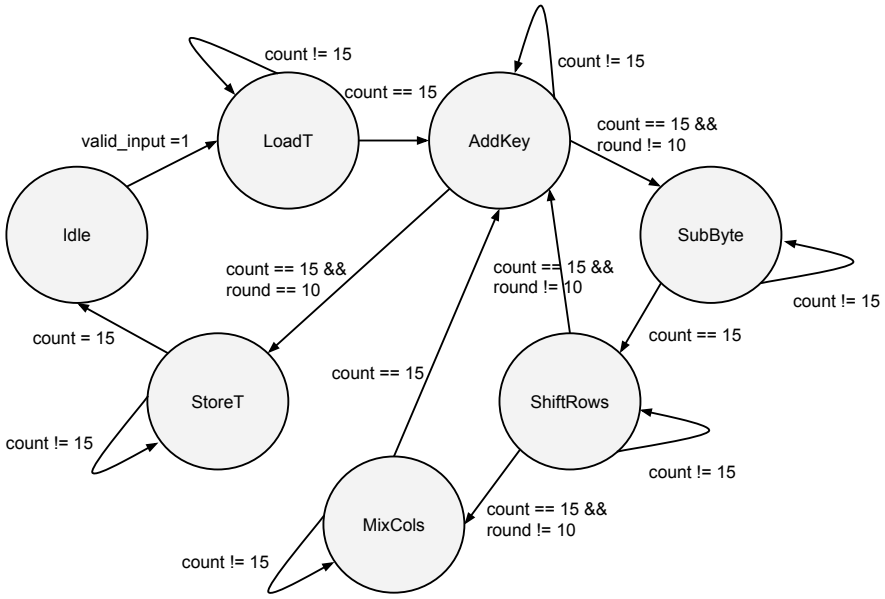


Figure 4.9: The state machine which controls the AES algorithm.

In the standard CMOS implementation the SubBytes operation is performed using a simple lookup table. This implementation is used as a baseline to perform DPA attacks. The proposed neuromemristive architecture uses the presented hardware neural network in place of the lookup table.

Chapter 5

Simulation and Attack Framework

This chapter discusses how the proposed architecture was simulated and attacked. In the simulation stage the architecture is validated for correctness and power traces are extracted. In the attack stage the power traces are processed along with the input data to the simulation.

5.1 Simulation Framework

5.1.1 AES Encryption Unit

The AES encryption unit was written in System-Verilog and the design is compiled and synthesized to a gate level implementation. The simulation is controlled by a testbench which specifies the input data, rounds of execution, and number of executions. A random plaintext is generated for every execution. Only the first round of the AES algorithm is simulated since attacks on subsequent rounds cannot be performed without knowing the secret key. Several thousand execution iterations of AES are needed to perform a successful DPA attack and power data from these iterations is gathered during a single simulation.

This simulation flow uses Synopsys tools for the compilation, synthesis, simulation, and power extraction of the design. The compilation and systhesis steps are shown in Fig. 5.1. The source files for the AES encryption unit are compiled together with the target implementation library using the Synopsys Design Compiler. The output is a single Verilog file that contains the gate-level implementation of AES.

The simulation and power extraction steps are also shown in Fig. 5.1. The compiled gate-level netlist and the testbench are fed as inputs to Synopsys Verilog Compiler Simulator (VCS) and to the PrimeTime power analyzer.

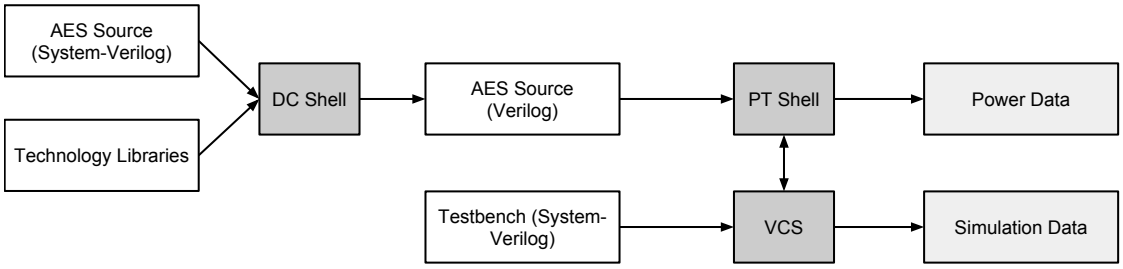


Figure 5.1: The source file and testbench are simulated and both the simulation outputs and power at each step of execution are gathered.

The outputs from VCS and PrimeTime are a *.out* and *.txt* file. The *.out* file contains the power consumption of the design for every event that occured during simulation. The timestamp in this file has a resolution of .01 ns. The *.txt* file contains the start time of each individual encryption along with the input plaintext and the output ciphertext. In this case the ciphertext is not a valid AES encrpytion output since only one round was executed, not ten. With these two files a DPA attack can be ran.

5.1.2 Neural Network

The neural network which performs the SubBytes operation was implemented in HSPICE. First the individual components of the design such as neurons and synapses were simulated independently and validated for correctness. The actual network itself was generated using a Python script which combined the individual components into a full network. This network is then validated to ensure it has the correct output values for every input combination. This process can be seen in Fig. 5.2. First the network is prototyped in Matlab and weight information is exported to a Python script. The Python script then generates the network and simulates every input combination through HSPICE. The outputs are then read out from a measurement file and the performance is checked in Matlab. If the network accuracy is less than 100% the network has go through further training iterations. If

the training process is complete the power data can be extracted using HSPICE's built in measurement tools.

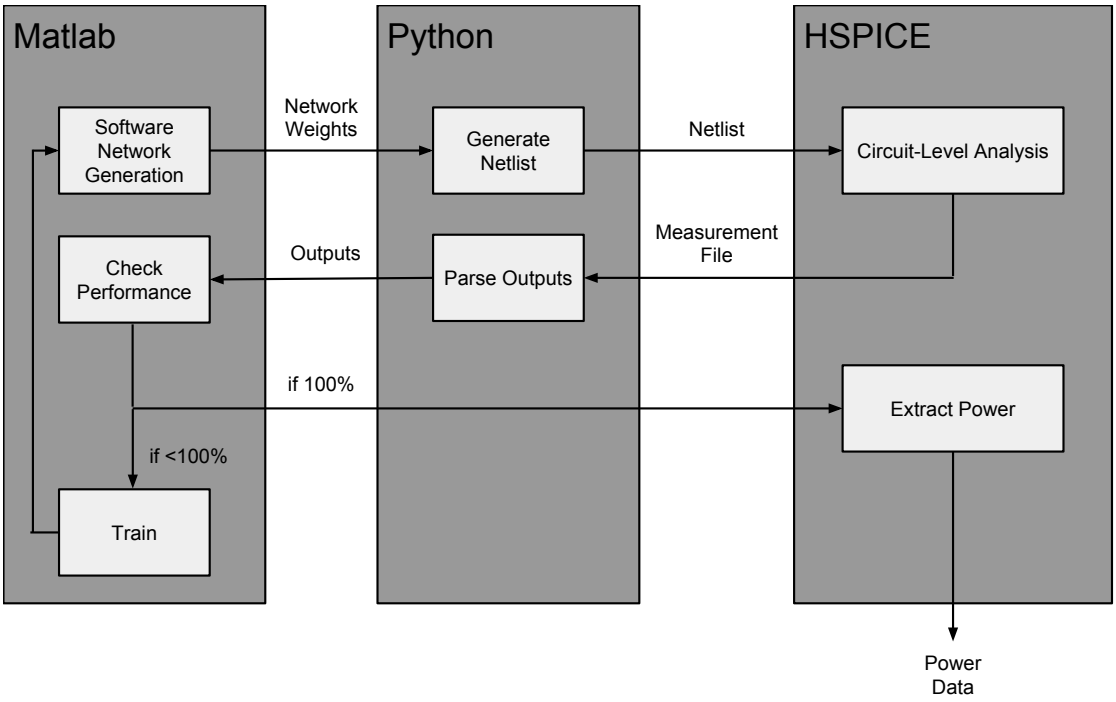


Figure 5.2: The process used to generate and test a hardware neural network.

The power data extracted from the HSPICE model had a resolution of .01 ns to match the resolution from PrimeTime. The two sets of power data can then be combined together to find the total consumption of the circuit.

5.2 Attack Framework

The attack framework has two main inputs, the power traces and the plaintext information. The DPA algorithm uses instantaneous power consumption of the system when the secret key was being used. Therefore, the algorithm first has to extract the relevant power consumption values from the supplied power trace files.

The particular operation being targeted in this attack is the S-Box lookup in the first round. Equation 5.1 shows the equation that represents this operation. In the case of a first round

attack only the plaintext is known. The Key byte has 256 possible values and this value is what the attack is attempting to guess.

$$ByteOut_i = SBox[Plaintext_i \oplus Key_i] \tag{5.1}$$

This operation will occur at a particular offset away from the start of the encryption. The calculation of $ByteOut_0$ occurs first, then $ByteOut_1$ and so on. Since the start time of each encryption is known the algorithm simply has to go through the power trace data until it arrives at the correct timestamp. Once the power trace for a particular operation has been found it is then sorted into a low or high bin depending on the particular key guess. There are 256 pairs of bins, one pair for each key guess. If the $ByteOut$ resulting from a particular key guess has more '1' bits than a threshold value then the trace is put into the high bin. Otherwise it is put into the low bin.

Once all the power traces have been sorted the average power consumption in each bin is found. The differential trace for each key guess is the difference in power between the high and low bin. If a particular key guess was correct, the differential trace for that guess will be greater than the differential trace for all other key guesses. The confidence ratio [40] is a metric for determining the likelihood a key guess is correct and is defined as the maximum differential trace divided by the next highest differential trace. A confidence ratio of 1 means that a key guess is not likely to be correct, and the confidence ratio generally increases as the number of traces gathered increases for a successful attack. The expression to find the confidence ratio is shown in Eq. 5.2.

$$C_k = \frac{max(\Delta_k)}{max(\{\Delta_0, ..., \Delta_N\} - \Delta_k)} \tag{5.2}$$

This framework was developed in Matlab. Once all of the necessary power traces have been found they can be sorted in parallel since the computation for a particular key guess is independent of all other key guesses. In the case of pure CMOS implementation the power traces are already in one file. When collecting power traces for the CMOS and

hardware neural network architecture the traces from both power collection methods have to be combined by matching timestamps. Once they are combined into one file the attack process is the same. The process is illustrated in Fig. 5.3. In this attack 8 bits are guessed at a time so the *Predict Byte Output* step requires 256 predictions. Each of those predictions need to be sorted independently so there are a total of 256 '0' bins and 256 '1' bins for each partial guess. Since there are 128 bits in the key there are a total of 16 partial guesses required to guess the entire key. Even if the attacker cannot correctly guess every key byte with high confidence every byte that is found makes a brute force attack easier.

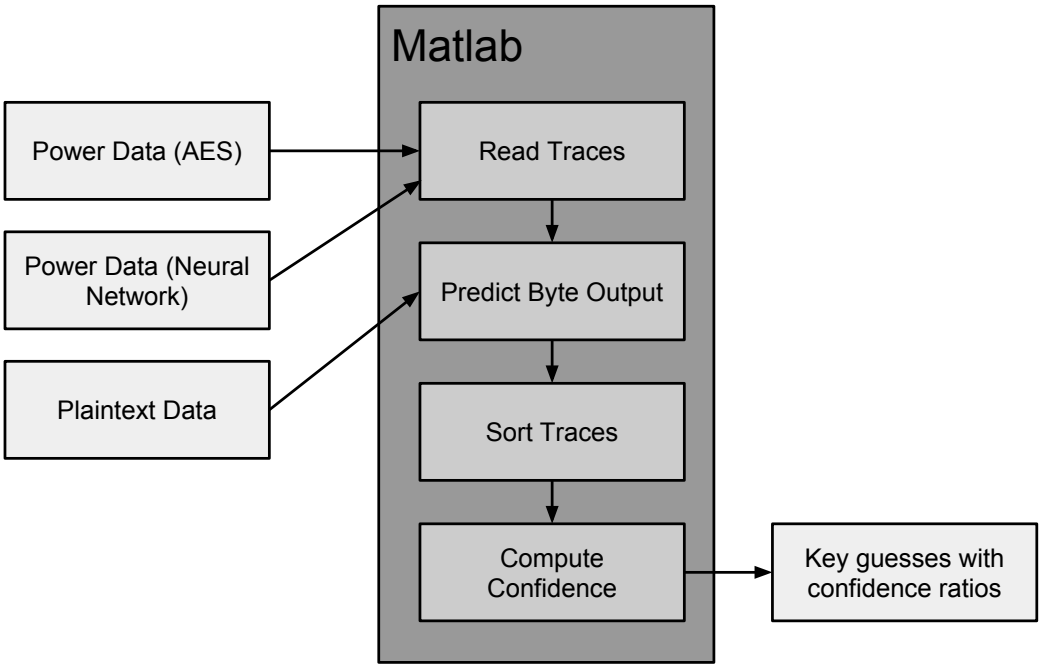


Figure 5.3: The attack process is implemented using Matlab scripts which take power and simulation data as inputs and yield key guesses for each byte and the confidence ratio for each guess.

Chapter 6

Results and Analysis

6.1 Neuron and Synapse Tests

A critical component of a neural network's ability to learn difficult functions is its activation function. A linear activation function is only able to learn linearly separable functions and is therefore unsuitable for the S-Box function. Instead a sigmoidal function was used that approximates the hyberbolic tangent function. A true hyperbolic tangent function is not necessary since problems such as a limited input range can be compensated for by the training process. Fig. 6.1 shows the simple framework used to test the synapse and neuron together. The weight values range from -1 to 1, and the input magnitude is generally swept to capture the full range of the activation function.

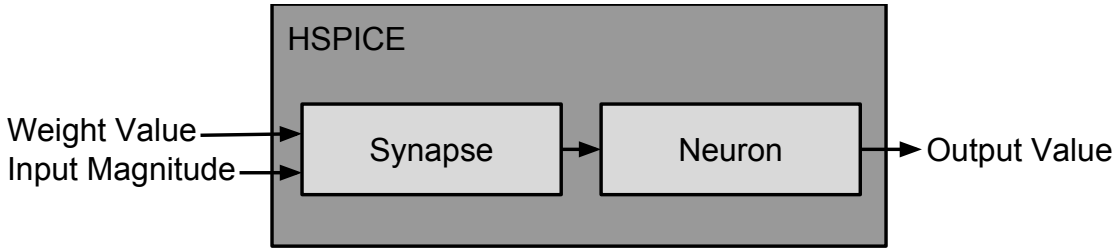


Figure 6.1: The framework used to test the neuron and synapse circuits.

The sigmoid function generated by the neuron circuit is compared to an ideal sigmoid in Fig. 6.2. This plot was generated using a synapse weight value of 1 and the input magnitude was swept from -0.5V to 0.5V. The ideal sigmoid clearly has a shallower slope and approaches its output limits more gradually. The circuit also has a more limited output range since its inputs range from -0.5V to 0.5V but the output is limited to -.35V to .4V. The

changes from the ideal sigmoid make it impossible for network weights to be transferred directly from software to hardware without losing some accuracy. In some applications a small loss of accuracy might be permitted, but in the case of the AES S-Box 100% accuracy is required. Even adjusting the models when training in software to mimic what is found in hardware led to little improvement in training results. One way the accuracy can be improved is by using extra neurons in the hidden layer so that the hardware could have an easier time converging to 100% accuracy.

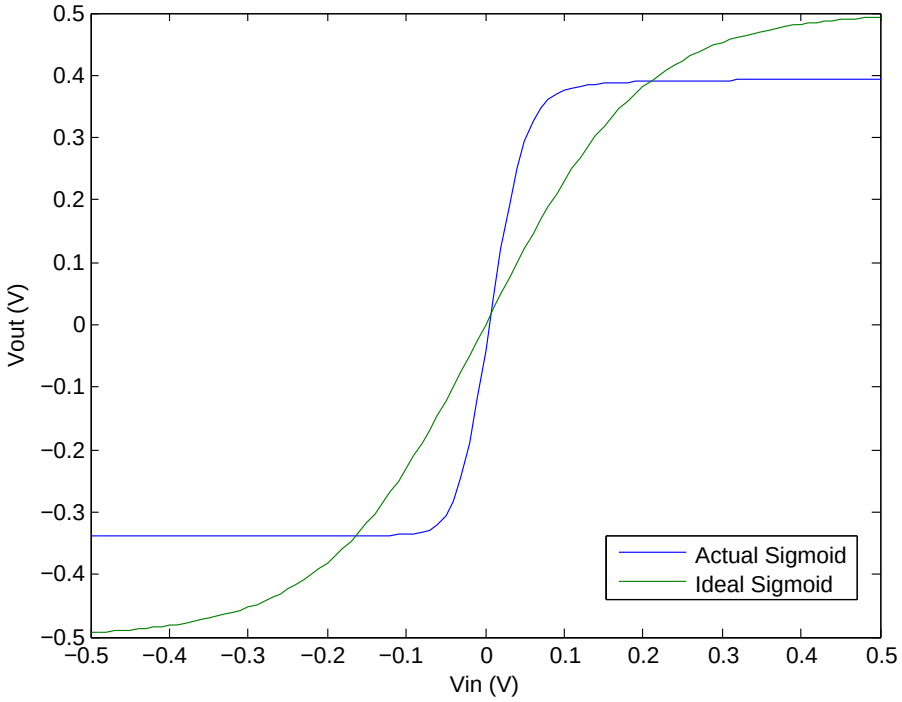


Figure 6.2: The ideal sigmoid shape and output measured from the neuron circuit.

The synapse circuits also had to be evaluated to ensure a wide range of weight values were available. Fig. 6.3 shows a few possible weight values. Both positive and negative weight values are shown. These weight values are selected by putting one memristor in its high resistance (off) state and adjusting the other memristor to the desired weight. For positive weight values the negative resistor is off and the opposite is true for negative weight values. A synapse weight of zero is made by turning both memristors off. There is still a small

negative output even with a synapse weight of zero but this is compensated for during training.

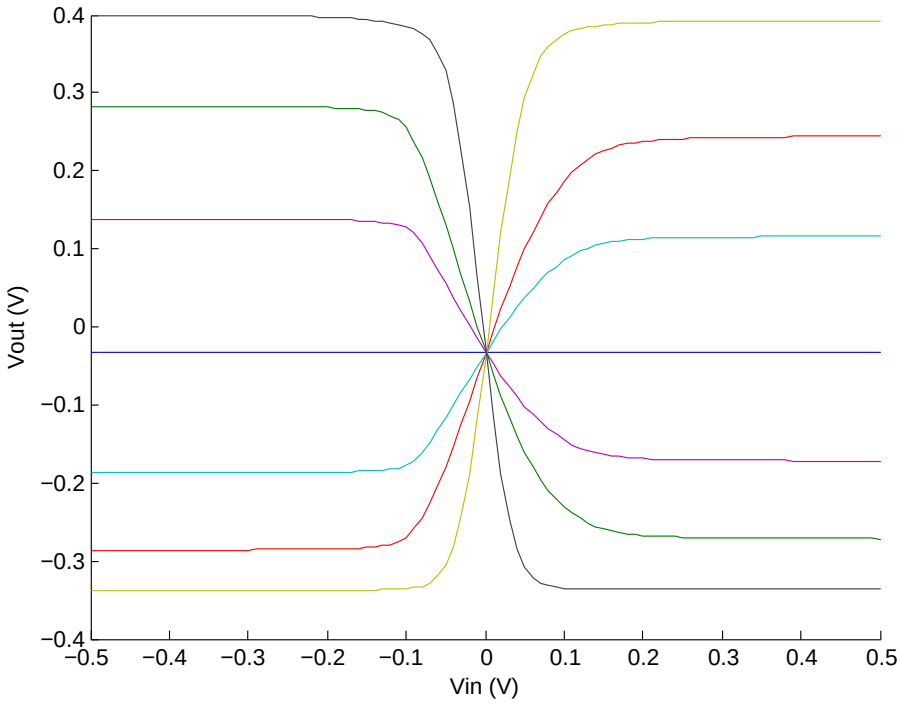


Figure 6.3: An example of different synapse weight values and their effect on the sigmoid activation function output.

6.2 DPA Attack Results

Three architectures were evaluated to determine their resistance to DPA. These architectures are shown in Table 6.1.

Architecture	**S-Box Implementation**
CMOS	CMOS LUT
CMOS + 8:48:1 ANN	8 × 8:48:1 ANN
CMOS + 7:48:2 ANN	8 × 7:48:2 ANN

Table 6.1: The architectures attacked using DPA.

The first system tested was the full CMOS system. This implementation was not resilient

to DPA which can be seen in Fig. 6.4 and Fig. 6.5. Fig. 6.4 shows the difference between the A_0 and A_1 bins as more traces are gathered. Each line corresponds to a key guess, so this figure only shows the guesses for one byte of the key. One differential trace is clearly higher than all the rest which becomes apparent after only a few hundred traces are gathered.

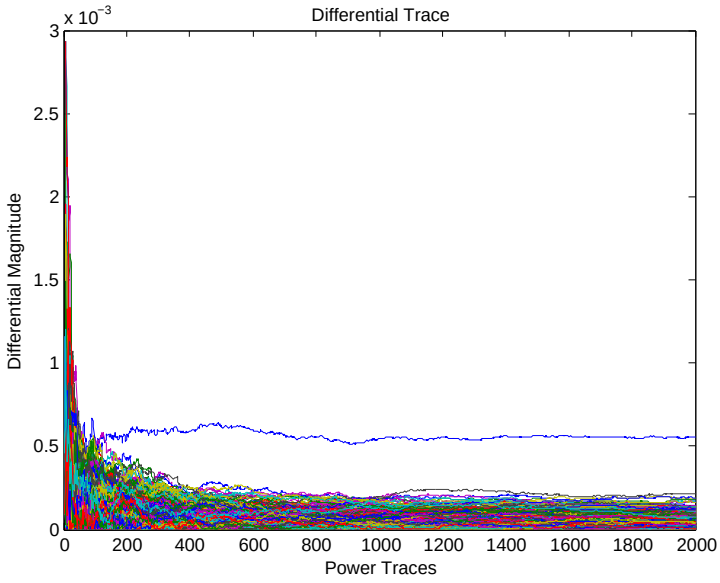


Figure 6.4: The differential traces for different guesses of the first byte of the key in a fully CMOS architecture.

Fig. 6.5 shows the confidence ratio for every key byte. After only 2000 traces are gathered the confidence ratio is above 2.0 for every byte. Each corresponding key guess was also determined to be correct. This shows that the CMOS architecture is clearly not secure against DPA. Multiple trials showed that the key bytes could consistently be guessed correctly without requiring an infeasible number of power traces. These attacks were done using a Hamming Weight leakage model.

Next the CMOS architecture using a hardware neural network for the S-Box function was tested using the same number of power traces. The power consumption of the neural network for every input combination is shown in Fig. 6.6. This data was collected from eight eight-input, one-output networks. The power consumption of the network is clearly related

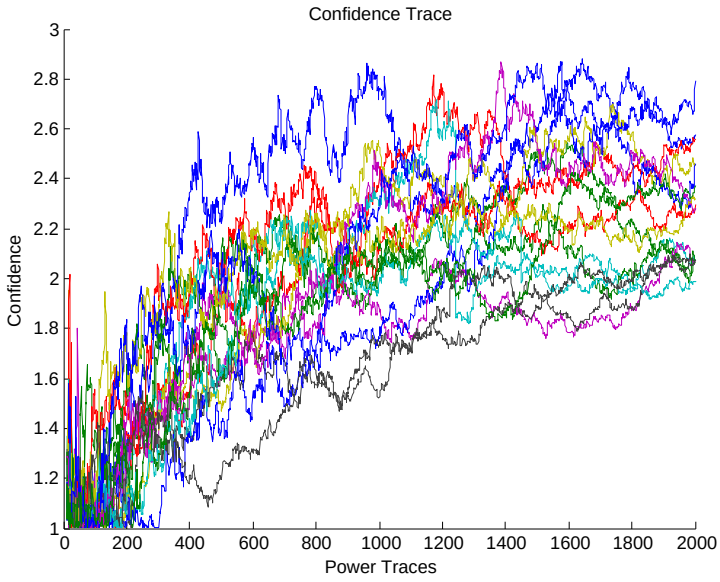


Figure 6.5: The confidence ratios for all key bytes from the fully CMOS architecture.

to the input value of the network. There are nine distinct power levels which correspond to 0-8 input bits being high. This means that the power consumption of the network is not dependent on the S-Box output value which implies it will not be vulnerable to DPA.

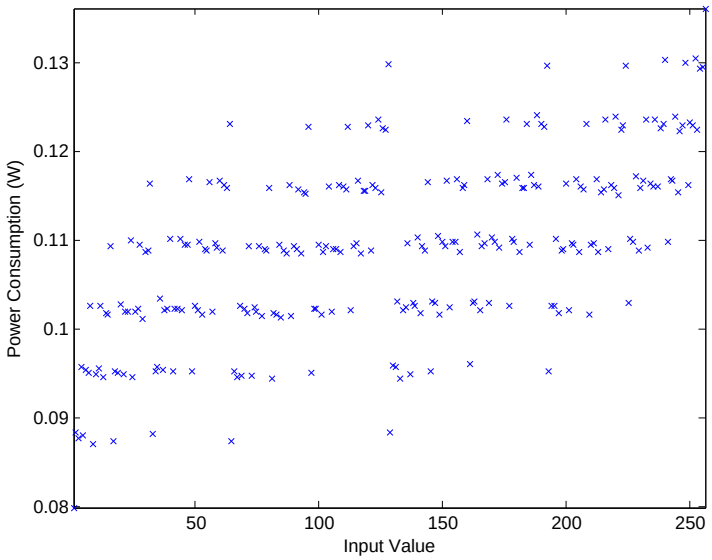
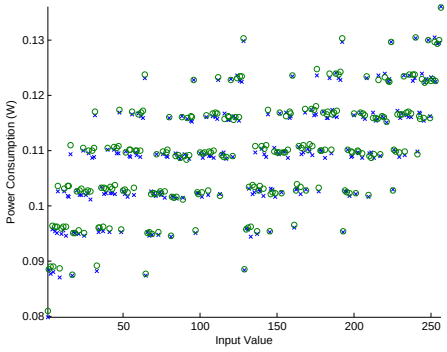
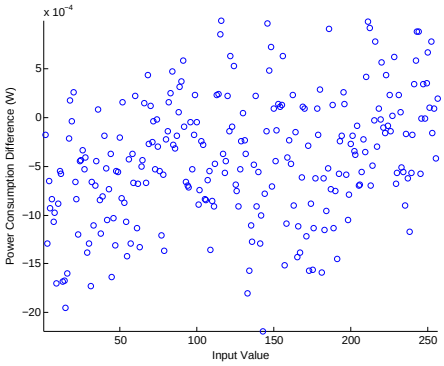


Figure 6.6: The power consumption of the network for every input combination.

Retraining the network also changes the overall power signature for each input value. In Fig. 6.7a the power consumption of two different networks are plotted together. In Fig. 6.7b just the difference in power consumption is plotted. The overall consumption is dominated by the input value, but the change of network weights still has some effect.



(a) The power consumption of two different 8:48:1 neural networks for every input combination.



(b) The difference in power consumption of two different 8:48:1 neural networks for every input combination.

In Fig. 6.8 it can be seen that no single differential trace was clearly higher across all the key byte guesses. This can be confirmed by looking at the confidence ratios in Fig. 6.9. After 2000 traces no confidence ratio is consistently above 1.4 and no successful key guesses were made.

The neural network architecture using random input values was also evaluated over 2000 traces. As expected, this architecture performs as well as the previous neural network architecture for 2000 traces. In Fig. 6.10 no differential trace is clearly higher and in Fig. 6.11 there are no confidence ratios above 1.4.

The neural network architectures are clearly more resilient than the pure CMOS implementation. However, a true attack may use several thousand more traces so next the networks were tested using 40,000 power traces. Figs. 6.12 6.13 shows the results of an attack using 40,000 power traces. Even with that many traces the attack cannot successfully recover a single key byte. All of the confidence ratios remained below 1.2 after some initial spikes in the beginning.

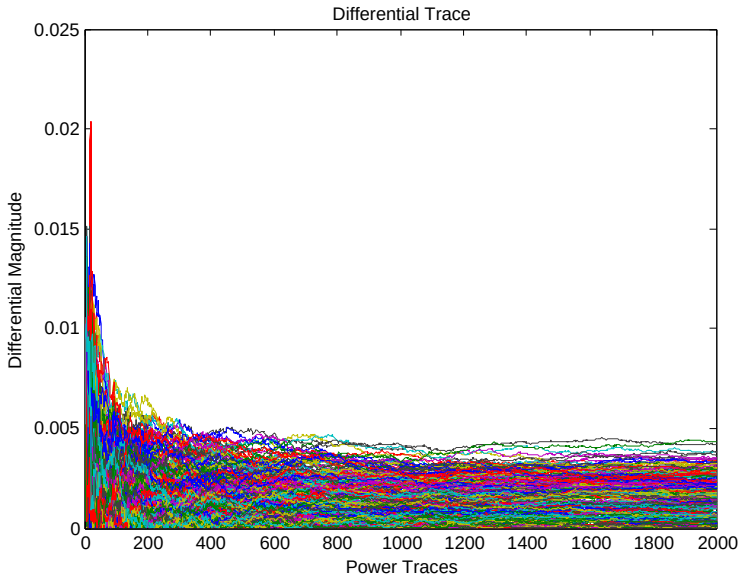


Figure 6.8: The differential traces for different guesses of the first byte of the key in a CMOS with the 8:48:1 neural network architecture.

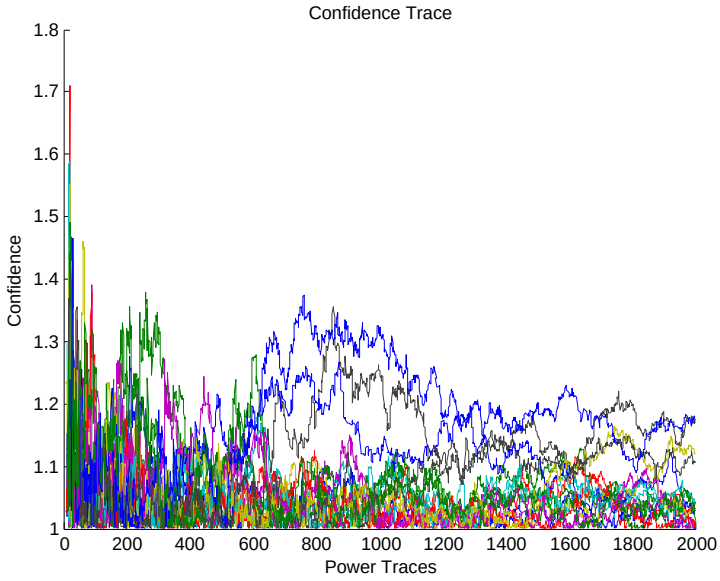


Figure 6.9: The confidence ratios for all key bytes from the CMOS with the 8:48:1 neural network architecture.

Next the neuromemristive systems were attacked using a Hamming Distance leakage model. This attack was more effective in that seven key bytes were correctly guessed. Despite these

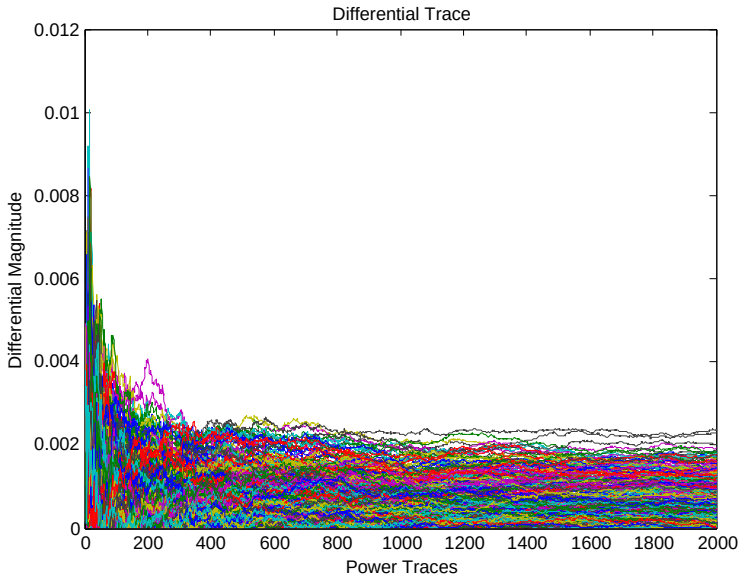


Figure 6.10: The differential traces for different guesses of the first byte of the key in a CMOS with the 7:48:2 neural network architecture with randomization.

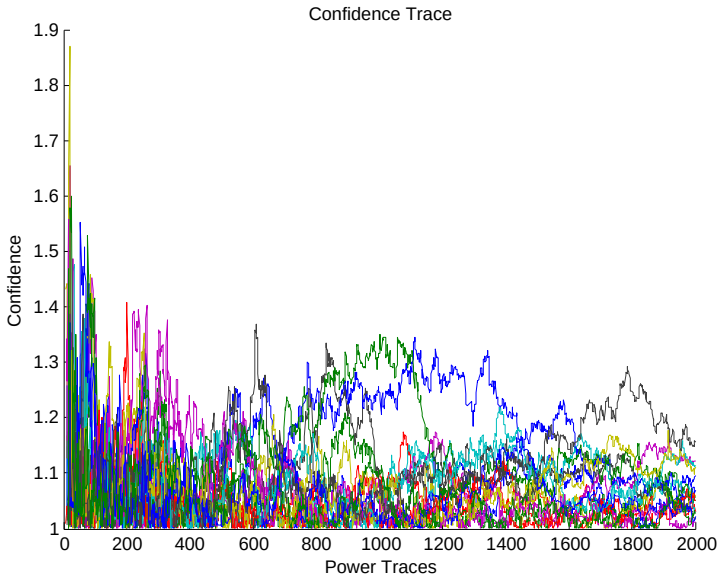


Figure 6.11: The confidence ratios for all key bytes from the CMOS with the 7:48:2 neural network architecture with randomization.

correct guesses the confidence ratio did not exceed 1.1 after 40,000 power traces as shown in Fig. 6.14a. In order to prevent the attacker from having any correct key guesses the

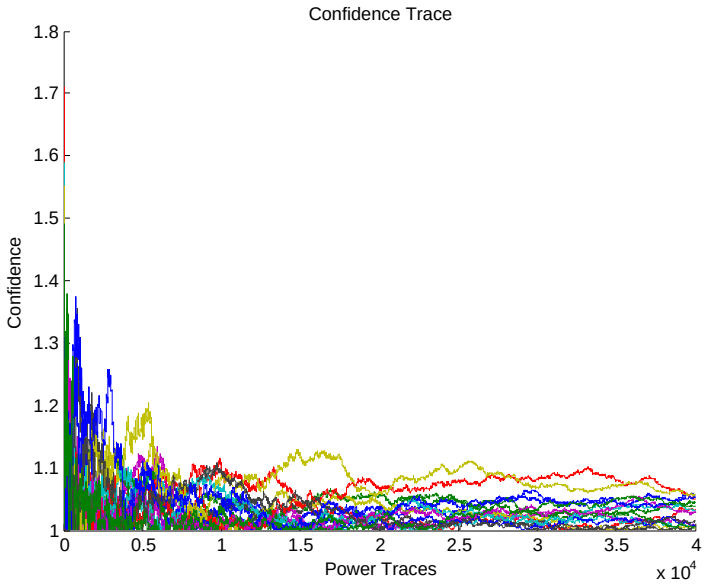


Figure 6.12: The confidence ratios for all key bytes from the CMOS with the 8:48:1 neural network architecture after 40,000 power traces.

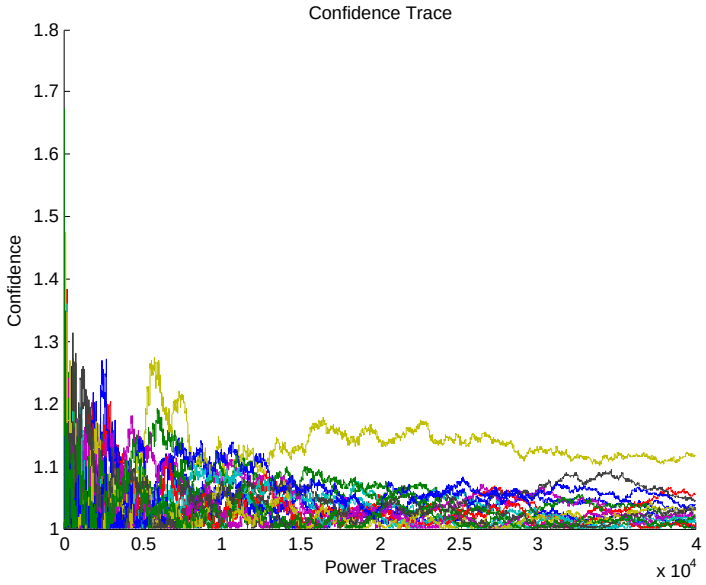
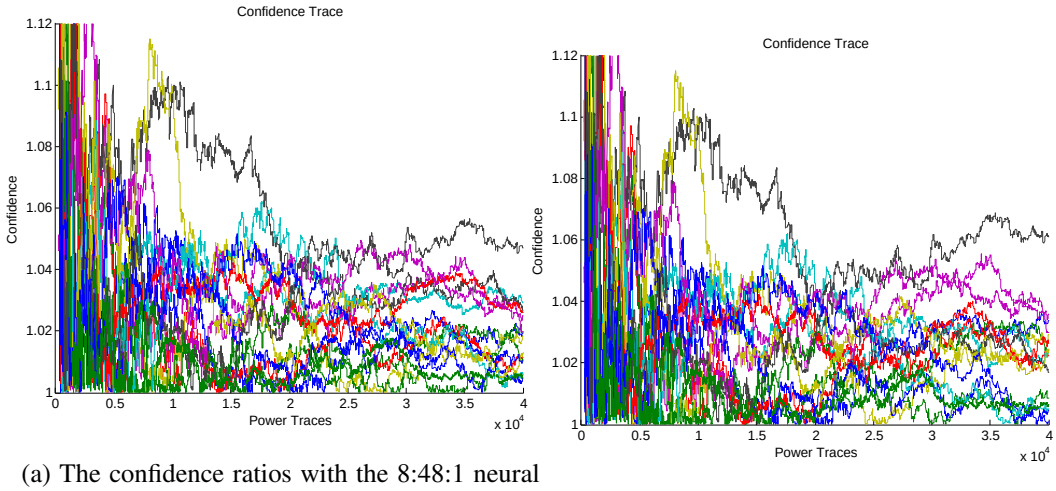


Figure 6.13: The confidence ratios for all key bytes from the CMOS with the 7:48:2 neural network architecture after 40,000 power traces.

network was retrained after 20,000 power traces to change its power signature. Fig. 6.14b

shows the confidence ratios changed slightly after retraining but all of the key guesses remained the same. This shows that the randomness gained from retraining the network is not large enough to completely disrupt an attack after a large number of samples have already been taken.



(a) The confidence ratios with the 8:48:1 neural network architecture with a Hamming Distance leakage model.

(b) The confidence ratios with the 8:48:1 neural network architecture with weight retraining.

An attack on the 7:48:2 architecture with the Hamming Distance leakage model was unable to successfully guess any key bytes. The confidence trace for this attack is shown in Fig. 6.15. No confidence traces were consistently above 1.1. The random inputs to the network were therefore able to stop this DPA attack.

The strong effect the input values have on the network power consumption may make it possible to attack the AddRoundKey transformation rather than the SubBytes transformation. In practice, the SubBytes transformation is attacked because it is non-linear which makes it more vulnerable to DPA [20]. However, this neuromemristive implementation was attacked at the AddRoundKey stage regardless to ensure its safety against DPA. As expected, the system withstood an attack after 40,000 power traces without any confidence ratio above 1.1 and no correct key guesses. The confidence traces from this attack are shown in Fig. 6.16

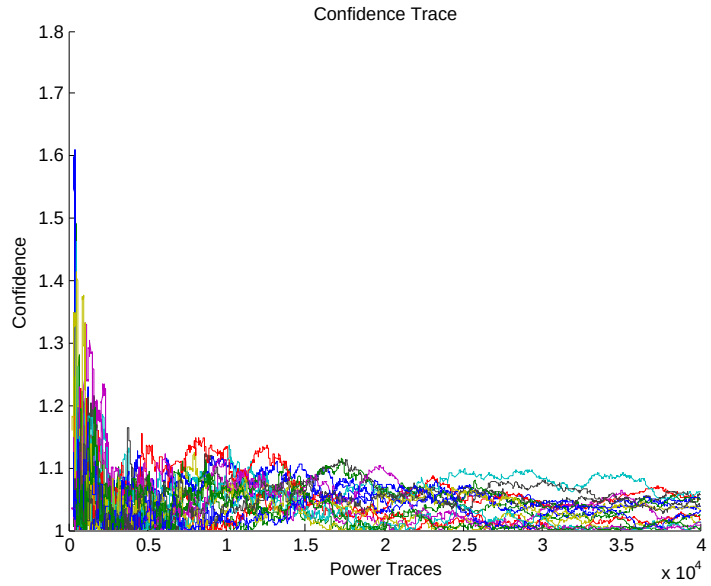


Figure 6.15: The confidence ratios for all key bytes from the CMOS with the 7:48:2 neural network architecture after 40,000 power traces and a Hamming Distance leakage model.

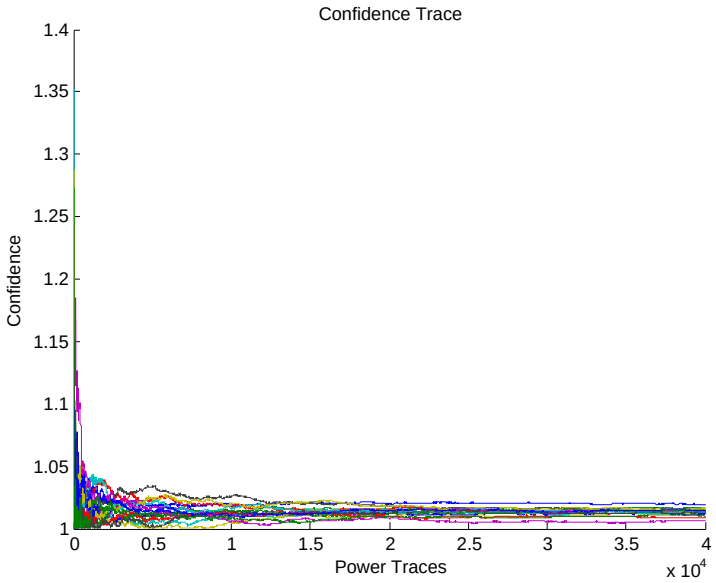


Figure 6.16: The confidence ratios for all key bytes from the CMOS with the 8:48:1 neural network architecture after 40,000 power traces attacking the AddRoundKey stage.

The main drawback of this neuromemristive implementation is its power and area consumption. With the pure CMOS implementation the AES circuit consumed power on the

order of approximately 10 mW. The power consumption of the neural network itself is on the order of 100 mW when producing a value. There are several factors which contribute to this high power consumption. One is that the network was designed using the 180 nm techonology node, while the CMOS library used for the AES design was at 120 nm. The size of the networks also is quite large. Table 6.2 shows the number of transistors and memristors needed to implement the neuron and synapse components. The overall number of components needed for different S-Box designs is shown in Table 6.3. The CMOS LUT assumes 6T SRAM cells and a total of $8 * 256$ bits are needed to store the S-Box.

Component	**Transistors**	**Memristors**
Neuron	11	0
Synapse	2	2

Table 6.2: The size of neuromemristive components.

S-Box Design	**Neurons**	**Synapses**	**Transistors**	**Memristors**
8 8:48:1 Networks	392	3456	11224	6912
8 7:48:2 Networks	400	3456	11312	6912
CMOS LUT	-	-	12288	-

Table 6.3: The size of different S-Box designs.

Overheads such as addressing logic and bias current sources were ignored when computing these values. Although the CMOS and neuromemristive designs use similar numbers of transistors, the neuromemristive design is analog and its transistors spend more time in the switching region. The relatively huge power cost of the neuromemristive design could be mitigated by power gating it when it is not in use since all of its ”memory” is stored in non-volatile memristors.

Another technique to use memristors to mitigate DPA was proposed by Khedkar [16]. In this approach RRAM was used to balance the power consumption of memory accesses. Using RRAM instead of SRAM for all the memory greatly reduced the average power of an AES encryption from 8 mW to 2 mW. The implementation was also resistant to DPA even after 40,000 traces had been gathered.

An adaptive masking technique was used by Smith [39] to protect AES from DPA. The

masking approach protected the AES implementation over 50,000 traces without a single correct key guess. The masking technique had small area and power overheads.

Chapter 7

Conclusions and Future Work

7.1 Conclusions

This thesis work proposed a technique to mitigate DPA using a neuromemristive architecture. A framework for building ANNs in hardware was designed using a combination of Matlab and Python. This framework was then used to train an ANN to compute the SubBytes transformation in AES. The power profile of these ANNs increased the resilience of AES to DPA compared to a baseline implementation. Synopsys tools were used to extract power consumption from the baseline AES implementation and HSPICE was used to simulate and extract power from the ANNs. An attack framework was created in Matlab which could run DPA attacks in parallel. This attack framework was verified to work against a pure CMOS AES implementation and then used against the neuromorphic architecture.

The attack framework was able to correctly guess all key bytes of the baseline CMOS implementation in less than 2000 power traces. DPA was less effective against the neuromemristive architectures. In the case of a Hamming Weight leakage model the proposed design was able to withstand attacks that used 40,000 power traces. When a Hamming Distance leakage model was used the network architecture without any additional randomization had several of its key bytes guessed correctly but not with a high confidence ratio. Also, retraining the network to change its power signature while it was being attacked had a minimal effect on the key guesses. The power consumption of the proposed countermeasure was up to 10x higher than the total system power even though it had a similar number

of transistors. However, neuromemristive architectures show potential in mitigating DPA attacks without having to explicitly apply masking and hiding techniques.

7.2 Future Work

One important part of hardware neural networks that this work did not investigate was the feasibility of actual on-chip training. Training a neural network in hardware is a difficult problem that was not fully addressed and a comparison between hardware and software training algorithms valuable to see what is feasible. Memristor fabrication is still in its infancy and the amount of variation between memristors is large. This implies chip-in-the-loop training is necessary unless the unique parameters of every memristor can be stored in software.

While this design was resistant to first order attacks, higher order attacks may be more effective. Other attacks such as correlation power analysis (CPA) could also be mounted. Attacks unrelated to power consumption such as timing attacks or hardware trojans present a similar threat to security.

The field of neural cryptography has yet to be explored using neuromemristive architectures. Although there are no official standards that use neural cryptography, those algorithms are based on neural networks and would be well suited for implementation with the proposed neuron and synapse designs. The transformations in AES were designed to be efficient in software and digital logic, which made them difficult to model using the proposed architecture.

Bibliography

[1] S.P. Adhikari, Changju Yang, Hyongsuk Kim, and L.O. Chua. Memristor bridge synapse-based neural network and its learning. *Neural Networks and Learning Systems, IEEE Transactions on*, 23(9):1426 –1435, sept. 2012.

[2] A. Afifi, A. Ayatollahi, and F. Raissi. Implementation of biologically plausible spiking neural network models on the memristor crossbar-based cmos/nano circuits. In *Circuit Theory and Design, 2009. ECCTD 2009. European Conference on*, pages 563 –566, aug. 2009.

[3] Jude Angelo Ambrose, Roshan G Ragel, and Sri Parameswaran. Rijid: random code injection to mask power analysis based side channel attacks. In *Proceedings of the 44th annual Design Automation Conference*, pages 489–492. ACM, 2007.

[4] Daniel J Bernstein. Cache-timing attacks on aes, 2005.

[5] Andrey Bogdanov, Dmitry Khovratovich, and Christian Rechberger. Biclique cryptanalysis of the full aes. *Advances in Cryptology–ASIACRYPT 2011*, pages 344–371, 2011.

[6] André Chanthbouala, Vincent Garcia, Ryan O Cherifi, Karim Bouzehouane, Stéphane Fusil, Xavier Moya, Stéphane Xavier, Hiroyuki Yamada, Cyrile Deranlot, Neil D Mathur, et al. A ferroelectric memristor. *Nature materials*, 11(10):860–864, 2012.

[7] L. Chua. Memristor-the missing circuit element. *Circuit Theory, IEEE Transactions on*, 18(5):507 – 519, sep 1971.

[8] L.O. Chua. The fourth element. *Proceedings of the IEEE*, 100(6):1920 –1927, june 2012.

[9] Jean-Sébastien Coron and Louis Goubin. On boolean and arithmetic masking against differential power analysis. In *Cryptographic Hardware and Embedded SystemsCHES 2000*, pages 231–237. Springer, 2000.

[10] J. Daemen and V. Rijmen. Aes proposal: Rijndael. In *First Advanced Encryption Standard (AES) Conference*, 1998.

[11] Elke De Mulder, Siddika Berna Örs, Bart Preneel, and Ingrid Verbauwhede. Differential power and electromagnetic attacks on a fpga implementation of elliptic curve cryptosystems. *Computers & Electrical Engineering*, 33(5):367–382, 2007.

[12] I.E. Ebong and P. Mazumder. Cmos and memristor-based neural network design for position detection. *Proceedings of the IEEE*, 100(6):2050 –2060, june 2012.

[13] Victor Erokhin and Marco P Fontana. Electrochemically controlled polymeric device: a memristor (and more) found two years ago. *arXiv preprint arXiv:0807.0333*, 2008.

[14] Sylvain Guilley, Laurent Sauvage, J-L Danger, Tarik Graba, and Yves Mathieu. Evaluation of power-constant dual-rail logic as a protection of cryptographic applications in fpgas. In *Secure System Integration and Reliability Improvement, 2008. SSIRI'08. Second International Conference on*, pages 16–23. IEEE, 2008.

[15] Sung Hyun Jo, Ting Chang, Idongesit Ebong, Bhavitavya B Bhadviya, Pinaki Mazumder, and Wei Lu. Nanoscale memristor device as synapse in neuromorphic systems. *Nano letters*, 10(4):1297–1301, 2010.

[16] Ganesh Khedkar. Power profile obfuscation using rrams to counter dpa attacks. Master's thesis, Rochester Institute of Techonology, 2013.

[17] DJ Kim, H Lu, S Ryu, C-W Bark, C-B Eom, EY Tsymbal, and A Gruverman. Ferroelectric tunnel memristor. *Nano letters*, 12(11):5697–5702, 2012.

[18] Kuk-Hwan Kim, Siddharth Gaba, Dana Wheeler, Jose M. Cruz-Albrecht, Tahir Hussain, Narayan Srinivasa, and Wei Lu. A functional hybrid memristor crossbar-array/cmos system for data storage and neuromorphic applications. *Nano Letters*, 12(1):389–395, 2012.

[19] Alexander Klimov, Anton Mityagin, and Adi Shamir. Analysis of neural cryptography. In *Advances in CryptologyASIACRYPT 2002*, pages 288–298. Springer, 2002.

[20] P. Kocher, J. Jaffe, B. Jun, and P. Rohatgi. Introduction to differential power analysis. 2011.

[21] Paul Kocher, Joshua Jaffe, and Benjamin Jun. Differential power analysis. In Michael Wiener, editor, *Advances in Cryptology CRYPTO 99*, volume 1666 of *Lecture Notes in Computer Science*, pages 388–397. Springer Berlin Heidelberg, 1999.

[22] Wei Lu, Kuk-Hwan Kim, Ting Chang, and Siddharth Gaba. Two-terminal resistive switches (memristors) for memory and logic applications. In *Design Automation Conference (ASP-DAC), 2011 16th Asia and South Pacific*, pages 217–223. IEEE, 2011.

[23] Stefan Mangard, Norbert Pramstaller, and Elisabeth Oswald. Successfully attacking masked aes hardware implementations. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 157–171. Springer, 2005.

[24] M.R.G. Meireles, P.E.M. Almeida, and M.G. Simoes. A comprehensive review for industrial applicability of artificial neural networks. *Industrial Electronics, IEEE Transactions on*, 50(3):585 – 601, june 2003.

[25] Thomas S Messerges. Securing the aes finalists against power analysis attacks. In *Fast Software Encryption*, pages 150–164. Springer, 2001.

[26] Thomas S Messerges, Ezzat A Dabbish, and Robert H Sloan. Examining smart-card security under the threat of power analysis attacks. *Computers, IEEE Transactions on*, 51(5):541–552, 2002.

[27] M.N.A. Noughabi and B. Sadeghiyan. Design of s-boxes based on neural networks x002a;. In *Electronics and Information Engineering (ICEIE), 2010 International Conference On*, volume 2, pages V2–172 –V2–178, aug. 2010.

[28] Sıddıka Berna Örs, Elisabeth Oswald, and Bart Preneel. Power-analysis attacks on an fpga–first experimental results. In *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 35–50. Springer, 2003.

[29] Dag Osvik, Adi Shamir, and Eran Tromer. Cache attacks and countermeasures: the case of aes. *Topics in Cryptology–CT-RSA 2006*, pages 1–20, 2006.

[30] Elisabeth Oswald, Stefan Mangard, Christoph Herbst, and Stefan Tillich. Practical second-order dpa attacks for masked smart card implementations of block ciphers. In *Topics in Cryptology–CT-RSA 2006*, pages 192–207. Springer, 2006.

[31] Elisabeth Oswald, Stefan Mangard, and Norbert Pramstaller. Secure and efficient masking of aes-a mission impossible? *IACR Cryptology ePrint Archive*, 2004:134, 2004.

[32] Elisabeth Oswald, Stefan Mangard, Norbert Pramstaller, and Vincent Rijmen. A side-channel analysis resistant description of the aes s-box. In *Fast Software Encryption*, pages 199–228. Springer, 2005.

[33] Yu V Pershin and Massimiliano Di Ventra. Spin memristive systems: Spin memory effects in semiconductor spintronics. *Physical Review B*, 78(11):113309, 2008.

[34] Thomas Popp and Stefan Mangard. Masked dual-rail pre-charge logic: Dpa-resistance

without routing constraints. In *Cryptographic Hardware and Embedded Systems–CHES 2005*, pages 172–186. Springer, 2005.

[35] Mathieu Renauld, François-Xavier Standaert, and Nicolas Veyrat-Charvillon. Algebraic side-channel attacks on the aes: Why time also matters in dpa. *Cryptographic Hardware and Embedded Systems-CHES 2009*, pages 97–111, 2009.

[36] Matthieu Rivain and Emmanuel Prouff. Provably secure higher-order masking of aes. In *Cryptographic Hardware and Embedded Systems, CHES 2010*, pages 413–427. Springer, 2010.

[37] G.S. Rose, R. Pino, and Qing Wu. Exploiting memristance for low-energy neuromorphic computing hardware. In *Circuits and Systems (ISCAS), 2011 IEEE International Symposium on*, pages 2942 –2945, may 2011.

[38] Andreas Ruttor, Wolfgang Kinzel, and Ido Kanter. Dynamics of neural cryptography. *Physical Review E*, 75(5):056104, 2007.

[39] Garrett Smith. Power analysis attacks on the sha-3 candidate grostl. Master's thesis, Rochester Institute of Techonology, 2012.

[40] K. J. Smith. Methodologies for power analysis attacks on hardware implementations of aes. Master's thesis, Rochester Institute of Techonology, 2009.

[41] Sedat Soydan. Analyzing the dpa leakage of the masked s-box via digital simulation and reducing the leakage by inserting delay cells. In *Emerging Security Information Systems and Technologies (SECURWARE), 2010 Fourth International Conference on*, pages 221–227. IEEE, 2010.

[42] François-Xavier Standaert, Sıddıka Berna Örs, Jean-Jacques Quisquater, and Bart Preneel. Power analysis attacks against fpga implementations of the des. In *Field Programmable Logic and Application*, pages 84–94. Springer, 2004.

[43] M. Stottinger, S.A. Huss, S. Muhlbach, and A. Koch. Side-channel resistance evaluation of a neural network based lightweight cryptography scheme. In *Embedded and Ubiquitous Computing (EUC), 2010 IEEE/IFIP 8th International Conference on*, pages 603 –608, dec. 2010.

[44] Dmitri B Strukov, Gregory S Snider, Duncan R Stewart, and R Stanley Williams. The missing memristor found. *Nature*, 453(7191):80–83, 2008.

[45] Kris Tiri and Ingrid Verbauwhede. Securing encryption algorithms against dpa at the

logic level: Next generation smart card technology. *Cryptographic Hardware and Embedded Systems-CHES 2003*, pages 125–136, 2003.

[46] John Viega, Matt Messier, and Pravir Chandra. *Network Security with OpenSSL: Cryptography for Secure Communications*. O'Reilly Media, Incorporated, 2002.

[47] Jun Wu, Yiyu Shi, and Minsu Choi. Fpga-based measurement and evaluation of power analysis attack resistant asynchronous s-box. In *Instrumentation and Measurement Technology Conference (I2MTC), 2011 IEEE*, pages 1 –6, may 2011.

[48] Qiangfei Xia, Warren Robinett, Michael W Cumbie, Neel Banerjee, Thomas J Cardinali, J Joshua Yang, Wei Wu, Xuema Li, William M Tong, Dmitri B Strukov, et al. Memristor- cmos hybrid integrated circuits for reconfigurable logic. *Nano letters*, 9(10):3640–3645, 2009.

[49] Chris Yakopcic and Tarek M Taha. Energy efficient perceptron pattern recognition using segmented memristor crossbar arrays. In *Neural Networks (IJCNN), The 2013 International Joint Conference on*, pages 1–8. IEEE, 2013.

[50] Chris Yakopcic, Tarek M Taha, Guru Subramanyam, Robinson E Pino, and Stanley Rogers. A memristor device model. *Electron Device Letters, IEEE*, 32(10):1436–1438, 2011.

[51] Y. Zhou and D. Feng. Side-channel attacks: Ten years after its publication and the impacts on cryptographic module security testing. In *Information Security Seminar (WS 06/07)*, 2005.

...